ICL

Software 2000: *The output of a forum* a View *sponsored by ICL and the Commission of the European Communities* of the Future

edited by **Brian Randell, Gill Ringland** *and* **Bill Wulf**

Preface

A workshop entitled "Software 2000 - a View of the Future", sponsored by ESPRIT and ICL was held from the 6th to 8th April 1994 at Hedsor House, ICL's executive training centre. The twenty-two participants (from industry, academia and the public sector) were from Europe, Japan and the US. A list of these participants, along with their biographies is contained in Appendix A.

Participants submitted position papers prior to the workshop, describing their views and areas of interest. These were distributed to all the participants before the workshop and are included in full in Appendix B.

The workshop was divided into two main plenary sessions and three working group sessions that, in parallel, discussed Environment, Business and Technology. The body of this report reflects the working group structure, consisting of a chapter on each of the three main areas.

The scope of each working group session was very large, as was the quantity of notes taken. Each main chapter therefore consists of material from the following sources, with no clear distinction between them in the report:

- Direct quotations taken from audio tapes. These were running for the entire workshop.
- Paraphrased comments from the participants, conveying the sentiment of what was said.
- Text taken from the position papers submitted before the workshop.

Each of the main chapters is structured along the lines that were defined by the respective working group. Linking text conveys the flow of the discussion, or areas where there was general agreement. This is not attributed to any one person, and is shown indented in the text.

We, the "rapporteurs", wrote the draft for the main chapters of this report in the 48 hours following the workshop, reflecting a tight publication schedule. This resulted in a first draft that we then circulated to all the participants for comment. The editors - Brian Randell, Gill Ringland and Bill Wulf - then provided the executive summary and reviewed the document as a whole.

Our model for the style of the report was largely based on that of the 1968 NATO Conference on "Software Engineering". As did the authors of that report, we have endeavoured to restrict our role to the selection and classification of material.

It is important to note that the views expressed by the participants are those of individuals, not the organisations with which they are affiliated.

The report was supplied as camera ready copy to the ESPRIT printers and produced by them.

Thanks to Lou Durrant for providing administrative support and Liz Donaldson for arranging the notebook and DTP systems. We would also like to thank all the staff of Hedsor House for taking care of us during the workshop and the weekend that followed.

John Fitzgerald, University of Newcastle
Alfie Kirkpatrick, ICL
Rogerio de Lemos, University of Newcastle
Sue Schreiber, ICL
Terry Schumacher, European Computer Research Centre, Munich

Hedsor House
10 April 1994

III/6055-94

Contents

Executive summary

Executive Summary

Introduction

This summary describes the output from a Workshop on " Software 2000 - a View of the Future" held from 6-8 April 1994. It covers the major discussion points and conclusions reached, and some directions indicated by the discussions on areas for research related to software.

The motivation for such a workshop was:

- the belief that a set of discontinuities will be hitting software - technology, techniques, skills and applications - in the next few years; and
- there did not appear to be any coherent external body of analysis or thought on these issues.

The plan was therefore to get an international set of people together, from industry, academia, suppliers and users to brain-storm the topic.

The workshop was co-chaired by Professor Brian Randell (University of Newcastle upon Tyne, UK) and Professor Bill Wulf (University of Virginia, USA), and sponsored by ICL and Esprit. The subject area of the Workshop was deliberately broad, and included issues such as economics and legal issues - not just technology issues such as software engineering theory and practice.

All invitations were to named individuals, and participants contributed on a personal basis. The participants came from Europe, US and Japan, and were from a wide range of backgrounds:

- Academic research groups
- Consultancies/Industry watchers (representing IT departments)
- Consumer and entertainment industry companies
- Desktop computer/workstation companies
- Industrial research laboratories
- Major system purchasers
- Packaged software producers
- Systems Integrators/Facilities Management companies
- Telecommunications companies
- Traditional computer companies

Before the Workshop, the 22 participants (see Appendix A) identified the major factors that they felt would be likely to affect the future of software. These provided a basis on which the "home team" (Brian Randell, Gill Ringland, Bill Wulf) suggested that the discussion at the Workshop divided naturally into three areas (though with some overlap), namely:

i) the environment - economic, social, technical and legal/ commercial factors

ii) the business - the size and shape of businesses incorporating software and the changes (discontinuities) affecting them

iii) the technology of software - where is it going, what are the emerging technologies?

The workshop itself divided into a mixture of plenary sessions, and parallel sessions in Working Groups on each of the three areas. The bulk of the Report is divided into Chapters on the above topics, based on the discussions in the sessions plus the papers submitted before the Workshop. It was written by the rapporteurs in 48 hours immediately after the Workshop, for which we owe them a great debt of gratitude. Later edits and revisions have been made by circulating this for each participant to comment, and incorporating comments into a version frozen at the beginning of May and printed for us by Esprit DGIII - to whom many thanks.

The format of the Report, and the planning of the Workshop, had parallels with the 1968 NATO Software Engineering Conference (which had Brian Randell as a prime mover). As then, the structure of each Chapter was determined by the Working Groups, and the structure was fleshed out from the discussions and the position papers.

This summary attempts to highlight and summarize some of the main topics and ideas covered at the Workshop. It has been written, to bring together the main threads of argument and discussion. These did not always neatly fit within the domain of a Working Group, but to a large extent the sections below reflect the Chapters of the Report as follows:

- The Information Highway (Chapter 2)
- Hardware Price/performance Improvements (Chapters 1 and 3)
- Environmental Trends affecting Software (Chapter 1);
- Programmers and Programming (Chapters 3 and 4);
- Software Engineering Issues (Chapter 4)
- Research Issues (Chapters 3 and 4).

The Report, however, contains many useful insights which cannot be reproduced in a summary. So the reader is encouraged to follow the flavour of the discussions by reading the entire report, with the above road map in mind.

The information highway

If there was one topic which was dominant at the Workshop, it was that of the rapidly developing global information infrastructure. One aspect of this, the "information superhighway", a unified networking of information, entertainment and embedded processors, has been widely hyped. Underneath the hype, the Internet network has spread rapidly. It is growing at 10-15% per month. It started as an academic network but is already the basis of many commercial enterprises, and has over 20 M users. One of the most startling figures brought out at the Workshop was that the annualised rate of growth of the World-Wide "Web" (the user friendly front end) connected through Internet is 300,000%.

The existence of the "information infrastructure" will facilitate, if not demand, information services that we are all probably too myopic to see now. As an example of how the global information infrastructure is already changing the way of doing business: parts of the UK Ministry of Defence no longer keep copies of information standards, because they can instead go and get them off the network as needed. And they watch with interest experiments which show that distributed applications launched across the net can tap into far more "idle" computing resource than they have in-house.

There are signs that the late 1990s may see a surge of growth; that networking could fuel the creations of new businesses, with an energy similar to other major industrial sea-changes. Current activity is focused around information services (e.g. home shopping, entertainment) and collaborations between film, video, cable suppliers and IT companies. These activities make

it clear that the information infrastructure is not just the wire and fibres. The information (whether static information held in repositories, or the dynamic information involved in communication, co-operation or commerce) is what is important.

In this environment, new classes of application emerge. For instance a number of new applications relate to exploiting the network's resources, such as intelligent agents used to retrieve information and filter it. It becomes more sensible to use intelligence to search unstructured information than for the information to be pre-structured, so the emphasis moves from database applications to search applications for a domain, all though there is still improvement to be made in database design. The emphasis also shifts from software engineering to information engineering.

The traditional static structuring of information, as used in some legal databases for example, has proved to be unhelpful for most enquiries. And access to information over the net is difficult because there is so much data: information is the connection between data and purpose, so that clients which search for information, carrying with them semantic intelligence, will be crucial. This is essentially the area of knowledge representation. It is a well established community that aims to supply tools to represent a domain of knowledge

All this creates opportunities for new industries, with low barriers to entry. Even small players can take a specialised niche product to a world-wide market - though the essential role of government in creating digitised information sources is recognised. For example, no company would digitise the information in the British Library. And the trend will be towards more added value in content and less in pure technology. For example, the revenues collected by the networked information provider DIALOG service are allocated 50% to the information supplier, 40% to the network operator and 10% to the telecomms supplier.

The technical achievement of connectivity with increasing bandwidth is making us ask: what new options do we have and what are the social and governmental issues? How will they be resolved? We are now looking at a world with high connectivity where you can essentially have all the media wherever you want it.

The network forces the use of de facto standards, so as to build systems which interface with those already running. It changes our assumptions about system design. What we want from systems sounds the same as before, but the meanings we attach to the words have changed, because we now have the widespread use of networks and availability of components. We can use intelligence on the network. We operate in a heterogeneous environment, with imperfect knowledge about the other players. System building will use modularity and interconnection rather than a single data or system model.

The existence of the global net affects our use of law. Our view of jurisdiction is based on place, i.e. on physical presence. So, in principle, events in Thailand are within Thai jurisdiction. But once information is on the network and reaches the UK or the US then the rules set in the US or UK, which are very similar, apply. While rules on patent and copyright are reasonably alike country to country, liability rules differ even, within the US, by State; as software, embedded or packaged, becomes more pervasive in society, the question of jurisdiction for liability will become more central until the rules are harmonized. The choice of a set of rules is crucial.

Another impact, as software becomes more pervasive, is that the public perception of its effective quality will become more severely critical, and the increasing computer literacy of future generations will feed into the legal and commercial worlds.

On the question of how the global network should be regulated, it was clear that the rules and ethos of the academic community, with information sharing as a central ingredient, are inadequate for the commercial world. The view was taken that the additions should be evolutionary and commercially driven, rather than mandated by government. These changes will be needed to cover new issues of management, commercial structure and security.

A similarity was seen with the way standards bodies are now beginning to work. The traditional formal standards process is not able to keep pace with the changes currently seen in information technology. The role of the standards process becomes much more of information exchange to

accelerate convergence of de facto standards, and evolution of practicable processes, rather than the slow, highly formal processes of the past.

In summary, the emerging global information infrastructure will have massive effects on software's status as a "special" business. It will affect how it is designed and distributed, and what forms of industry will evolve in response to these changes. However, predicting just what the consequences will be is only possible at this stage in outline.

Hardware Price/performance Improvements

It is well understood that the cost of providing networking (telecoms) and of hardware processing & storage is continuing to decrease.

These trends have been visible for two decades and no slackening of the pace of change is seen. The net effect of these changes has been revolutionary rather than evolutionary in its effect on the economics of the supply industry - as evidenced by the dramatic changes in IBM's fortunes. Network technology will also continue to leap dramatically in performance and decrease in cost (even if this is not always reflected in prices in all regulatory or commercial environments - where the ability to provide a connection may provide a de facto monopoly).

On the positive side, technological innovation also puts vast processing power affordably on the desktop (now) or whatever the TV will evolve into (in the future), which can reach out into new applications and users. Another consequence is that many more products, especially those in the consumer electronics sector, will have increased functionality implemented by embedded software. Already, today, there is 0.5 Mbyte software in an expensive TV and 2Kbytes software in an electric shaver.

The use of processing power to make computers human literate rather than expecting humans to become computer literate, by using techniques such as visualisation and virtual reality, has hardly been exploited as yet. To do so will require skills in design, presentation and communication that have not typically been found in the software professionals, except perhaps in the games industry.

The plunging price for the storage of digital information is causing its use to soar, as many industries switch to digital property. Examples are publishing, films, newspapers, engineering drawings, maps. But the basis of ownership of information, intellectual property protection, is different when information can be replicated without trace, and its source is not apparent, i.e. physical presence cannot be used as the determinant.

The issue of how to treat digital property in law is in danger of being left to the lawyers. The lawyer participant at the Workshop was adamant that the software community needs to take a much more active role, to prevent unfortunate precedents being set. An example has been set recently in the "look and feel" cases, where US judges have appointed advisers knowledgeable on technical areas.

Environmental Trends affecting Software

The Workshop discussions identified a number of other trends which it was suggested seem certain to have significant effects on the software industry (or industries). These included:

- The world economic balance is changing, with the growth of Asian and Latin American economies. In the developed countries, the demographic shift to an ageing population gives different requirements for leisure, travel and health care.
- The role of the large organisation, world-wide, is expected to change; fewer core staff will be employed for a shorter "standard" working life, with most people working for smaller specialist companies with lower job security. The technology (networks, mobiles, desktops, information services) to support these companies will redefine business computing parameters.
- The shape of the IT industry is changing, with the advent of new competitors. The telecoms companies, cable, satellite, information and games companies, all package IT services differently, mostly for the home, but also for business.
- Business desktops in North America, Japan and Europe are rapidly becoming saturated. Replacement markets will be driven by new applications, and so the nature of the marketing and support effort - and their inter-relation - changes.
- The industry is expected to continue its divergence into organisations, some of which are customer facing, and some of which are product/distributor facing. A new layer of information provider is being added to the traditional services (network operator, and so on).
- The software developers' environment: is evolving fast. Technology for development will allow construction of larger projects, artificial intelligence (intelligent agents, knowledge based systems, data mining and intelligent filtering, and so on) will be increasingly feasible as costs decrease, performance improves and widespread networking is available.
- Prices for software components are dropping, and will continue to drop. Eventually basic level application components will be provided almost exclusively pre-packaged with system software at very low prices; increased bandwidth will allow networking of multimedia applications.
- Growth in the user population will include many novices (computer illiterates), who require simplified interfaces and few choices. New technologies (large high definition screens, voice, pen, scanning, and so on) give opportunities for creating an effective end user environment. Increasing desk top and embedded MIPs provide an opportunity to add intelligence.
- The increased computer literacy of the generation playing computer games at home and required to provide a PC for college studies may well change the definition of "technophobe" and end user, giving the software professionals a new set of concerns.

Programmers and Programming

There is a spectrum of types of programmer - from people who are very much professional programmers and know it, and who are paid by people who know it, through to people who do not regard themselves, and are usually not regarded by anyone else, as programmers.

In considering the programming task in general, several Workshop discussions centred on a phenomenon related to the development of programming, namely the notion of cycle and progression. We start from a system which is rigid in what it can do. Very often flexibility is then provided by allowing parameterization or customization, leading to a more complex system that is eventually recognized as needing new abstractions. A new abstraction is found, perhaps incorporating domain specific knowledge. Sooner or later, this extended in its turn, and the evolutionary process goes on.

As each discipline develops special techniques and tools, what people in each domain term s programming - whether the COBOL world or in-car intelligence -has become more different. Intellectually it is the same type of activity but technically it is different. This causes a fragmentation and spreading of the activities performed by programmers.

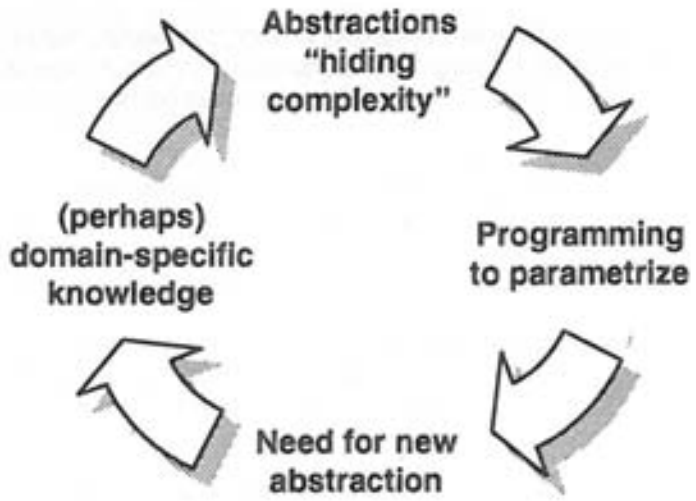

Figure E.1. Evolutionary cycle.

The academics, in considering education and training needs, differentiate between software professionals (the people who build underlying systems) and the professionals in some other domain who use basic software components to solve problems in their area, be it choreography or architecture. The number of people who design programs that will be used by somebody else is much smaller than the number of people who are involved at some stage in programming.

Professional programmers increasingly work outside the IT industry and IT departments. For instance, increasing processing and storage capability within a small size and low power requirements are revolutionizing the use of embedded software. Sewing machines are now optionally provided with ports for downloading programmes to control their stitching. And in HDTV the software may become effectively unbundled. The number of people involved in writing software for these embedded environments is growing amazingly fast, for instance the amount of code in consumer products is doubling every year. The question was asked - were

these programmers facing the complexity problems of the 1968 Software Engineering world all over again?

The evidence seems to be that, on the whole, the lessons of OS/360 have been learnt. It may well be that a large number of domain-specific software frameworks emerge for embedded environments: communications, entertainment creation (authoring), visualization and information retrieval. These would consist of applications programming interfaces, languages, tools, and so on. They would evolve and adapt very quickly, and each give rise to a software industry.

Software prices have fallen, ever since unbundling. The trend has been accelerated by the switch into desktops (and games), away from mainframes. New styles of distribution and support operate for this software. There are knock-on changes in IT departments, instrumentation, defence and consumer device companies, and the services industries, where the decreasing price of software components puts bespoke prices under pressure.

The effect on systems integration businesses based on projects to deliver new systems has also already been significant. Some of the US majors have had to lay people off because of the trend to use large numbers of software components and the effect of that on project price. The comparison of a custom system and a customized system based on existing components, in price and risk, has completely changed.

The lower cost of software components is also a problem in the software component industry itself. It must lower the capability for research and development. And it is hurting startups, who find that the cost of marketing even existing components they have on the shelf is too high to be worth it.

At the Workshop we introduced the term "intermittent programmer" to describe the relationship of "other professionals" to IT systems, applications, software and information. The term is meant to suggest that their use of a given set of software components is likely to be intermittent. We were clear that they need to deliver high quality output in their domain as much as software professionals in theirs', so that they need a far higher standard of usability, predictability, on-line training, and resilience for tools and components intended for their use.

We tried to quantify the sizes and growth trends of the four types of "programmer" identified -

- software professionals (in software companies)
- software professionals (in IT departments)
- software professionals (in other industries, producing embedded software)
- intermittent programmers.

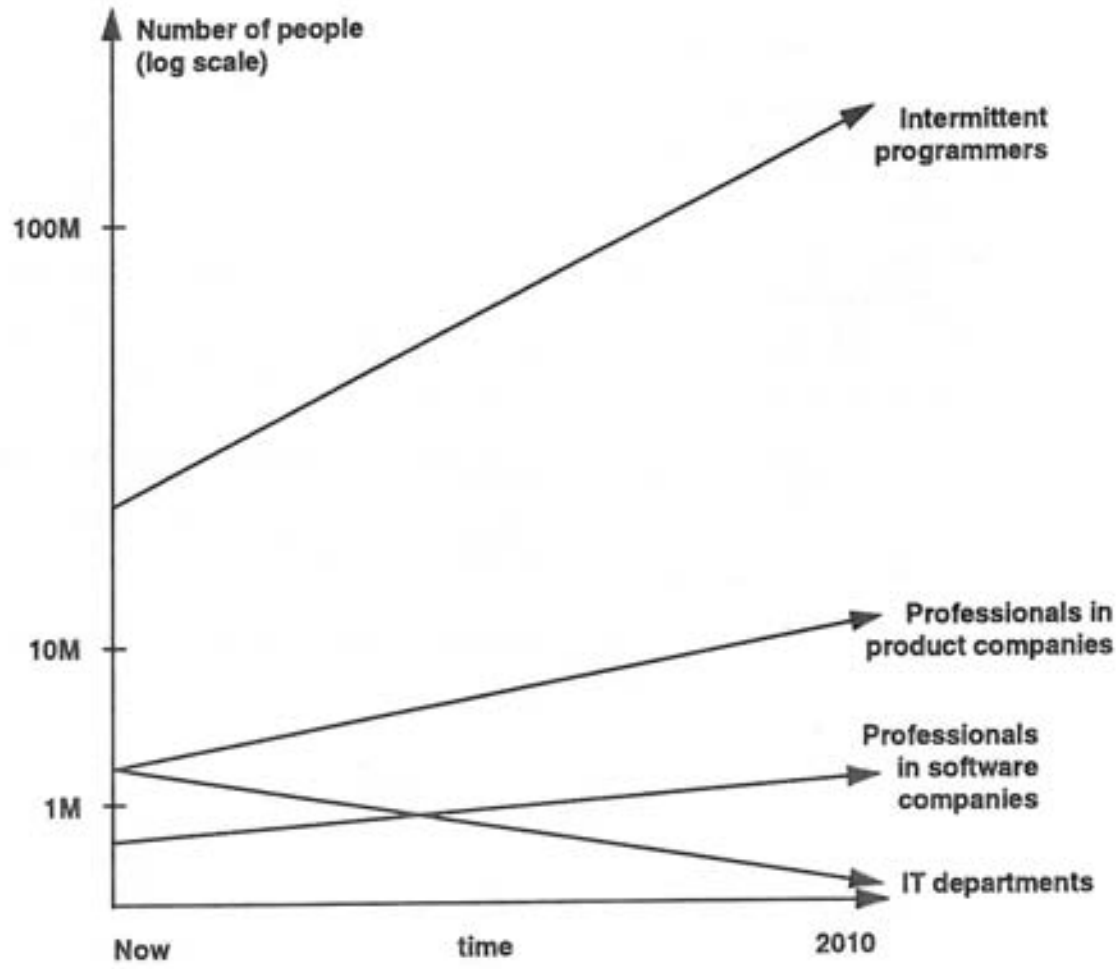

Figure E.2. Who programs when?

A model that may suggest what we are thinking is mathematics. There are few professional mathematicians in the world, yet each of us uses mathematics everyday and usually with out thinking 'now I'm doing math'. In the future, professional people in most fields will use programming as a tool, but they won't call themselves programmers or think of themselves as spending their time programming. They will think they are doing architecture, or traffic planning or film making.

This provides a model of the trend point for programming skills in the 21st century. In the same way that mathematics is ubiquitous, software is rapidly becoming so.

And, it also suggests that, although for the next few years the trends for professionals may be as shown, the effect of improved technology and re-use may start to limit the growth in the number of professionals needed within the next decade.

The Workshop also discussed the size of effort - in terms of man-hours - spent working in the four domains. The large number of intermittent programmers, even if they only currently use IT for 10% of their working time, represent more usage than the software professionals in IT departments and software companies combined. The effect of increasing their productivity is therefore very significant. In assessing the needs of these intermittent programmers, the view was expressed that users from the humanities provided interesting paradigms for "everyman".

Software Engineering Issues

For traditional computing systems, the problems of the magnitude worried about at the original NATO "Software Engineering" Workshop in 1968 can probably be handled today. We can build a system of the functionality of OS/360 without stumbling too badly, and compiler software is production-line.

A measure of progress is provided by the following non-exhaustive list of recently developed or recently successful software innovations:

- Object oriented programming (OOP) which integrates the notions of encapsulation, polymorphism and inheritance.
- Remote procedure call (RPC) and what it has enabled, e.g. client/server architectures.
- Application programming interfaces (APIs)
- Effective ad-hoc standards, e.g. Standard Query Language (SQL).
- High level languages applied to modelling, e.g. logic programming and constraint programming.
- Program transformation.
- Source level debugging.
- Multiple instruction multiple data (MIMD) in parallel computing.
- Graphical user interfaces (GUI) and GUI development tools.
- Advances in decompression and compression.
- Emulation improvements.
- Formal methods, automatic verification and theorem proving

However, despite developments like these, we are clearly not at a point where we can be satisfied with our ability to produce software that is reliable, on time and budget, cost-effective, user-friendly and so on. In short, software which is "fit for purpose".

Issues on the front burner today encompass the whole system view, so that the entire process of requirement capture, analysis and validation has become the concern, rather than capability to deliver to specification.

The usability of a system is today recognised as being as important as its functional correctness. There are many examples where the whole system, including the people, failed, not because the software per se failed, but because people incorrectly interpreted what the software was saying. Increasingly the expectation of software is that it will "do what I mean, not what I say".

On the software development process, the early process models are being abandoned as being too naive. For instance, the waterfall model does not support validation until late on in development. The automatic generation of the final program from a prototype is a direction which was seen to provide a framework for re-use. The increased reliability of today's systems is mostly due to informal techniques such as beta testing. The Chicago system, from Microsoft, for instance, will be tested by 10,000 people. The 48 Mbytes of the full Word 6 was developed without formal methods but with fault tolerance and extensive testing.

Re-use of software components was seen as critically important -(though this is not controversial !)- but the role of objects was seen as a step on the way rather than a complete answer. Advances to define and implement abstractions which operate in a distributed, heterogeneous environment, and which separate intent from implementation, are needed and

identified as a research area. The problems are those of how to bridge the inherent mismatch between the goals of the producer - who wants to be general to increase his market - and the consumer - who wants something highly specific for his environment.

The view was put forward that greatly improved levels of component re-use will ensue once we can create new abstraction mechanisms that are combined with specialization mechanisms at the same time. Every time something is abstracted, there should also be a mechanism to specialize it into either domain specific knowledge or to obtain performance. The use of subroutines is an example that always goes in one direction, always to abstraction. Once the technical obstacles are removed, by employing for example the meta-object protocol and similar specialization mechanisms, it was claimed that software re-use will grow enormously. This view was vigorously challenged on the grounds that the obstacles are not really technical, they are more to do with human nature than any particular software mechanism.

There was however no dispute over the view that the size and complexity of some software projects will always grow to the limits (or beyond) of our abilities to manage them. The major problem is often in capturing the requirements and delivering the system in a shorter timescale than they change. In spite of improvements in application building tools, there will still be a proportion of IT endeavours which stretch our capabilities to the limit. Businesses will always want some more ambitious IT system to give them competitive edge. The difficulties that beset these projects now: fluctuating and conflicting requirements, communication and co-ordination problems, will be just as prevalent in 10 years time. The organisation of teams across sites and geographies should be easier than it is: but human nature is such that even with teleconferencing and groupware, human interaction remains an essential part of the software development process.

This provides a guide to why, when software costs nothing to ship to (or from) the Far East, very little software is imported from there into the US. Software which is well specified can be made anywhere because the specification can be shipped, but in most cases the user interfaces in particular are not clearly enough defined.

Software implementation problems change in the age of the global information highway.

The changing balance of price/performance, between networking, disk storage and main memory, is changing the software designers' world. A lot of the programming structures that we have at the moment are essentially legacies of the fact that we have main memories and disks, and we have built up a world of databases and programming languages and all sorts of dichotomies that no longer make good sense.

And there is a huge transition when you go from addressing physical locations to addressing individuals as they adopt a mobile life style. The need to "graunch" (RAF slang for "to fit by use of excessive force") sets of legacy components within a heterogeneous network without degrading the overall system is a task to stretch the best integrators. And the problem of making controlled changes without stopping the world, is outside the experience of most systems.

There is vital legacy from mainframe approaches which is not being appreciated in the "new" environments. The discipline, operational standards, coherence and systems reliability assumed in the traditional environments need to be translated. This is not trivial - it is easy to make a central system twice or twenty times redundant, but nonsense to provide such redundancy independently for each of (say) two thousand times workstations.

Similarly, the task of software design and system implementation to align with business need inside a corporation is made more challenging as the business environment continues to be volatile. There have been several examples of companies who have had to "de-announce" mergers following the analysis of the IT systems as incompatible. The task of system integration was thought by the Workshop to be ripe for the application of artificial intelligence, to support the looser data and information connections needed for flexibility as organizations change their shape.

Trends in software engineering overall, even before we consider the revolutionary effect of the changes in the information highway, price/performance, and the software/programmer ubiquity, mean that at each stage there are a group of people who cannot progress from one

generation to the next. In the same way that some assembly language programmers were made obsolete by the introduction of higher abstraction level languages like FORTRAN and COBOL, and others found the next transition to a client/server PC based model difficult, transition to the software future will not be easy for many of today's software professionals.

Research Issues

Research Roles

The Workshop identified that there are several roles in research related to software.

In the classic model, such as might be found in chemistry, new ideas originate in the academic sphere then migrate to industry where they may be implemented, i.e. scaled up into industrial size process and introduced to the market. That model does not seem to reflect much of what happens in software. Examples include relational databases originating from Ted Codd at IBM, and the seminal role of Xerox Parc.

Research which experiments and evaluates tends to be driven by industry (or government) for two main reasons: it is risky - a realistic estimate based on venture capital statistics would be that 9 out of 10 projects "fail" in that they do not lead to innovation i.e. introduction to the market; and they tend to require larger, often multi-disciplinary projects. The role of academics as skilled partners in these projects is often crucial, though a different role from traditional academic research.

However, the concept of "failure" is relative. The work at Xerox Parc was a commercial failure for Xerox, and the first Apple systems incorporating the technology also failed. Nevertheless the work has been immensely influential in formulating the PC interfaces in use today.

It is difficult in many industrial organisations to manage a high level of experimentation, both culturally and financially. Culturally because bureaucracies act to try to avoid failure. Financially because the hardware vendors have traditionally funded this research out of their hardware margins, which have shrunk. The venture capital industry is not developed in Europe to the same extent as in the US, so this is a crucial question in Europe.

Potential drivers may come from other industries that are developing new businesses incorporating IT. They may be less IT literate and comfortable with IT.

This increases the need for traditional academic research in its roles of observation, analysis, and generalisation of concepts. In particular, without extensive involvement by academics it is less likely that technology transfer between domains will occur.

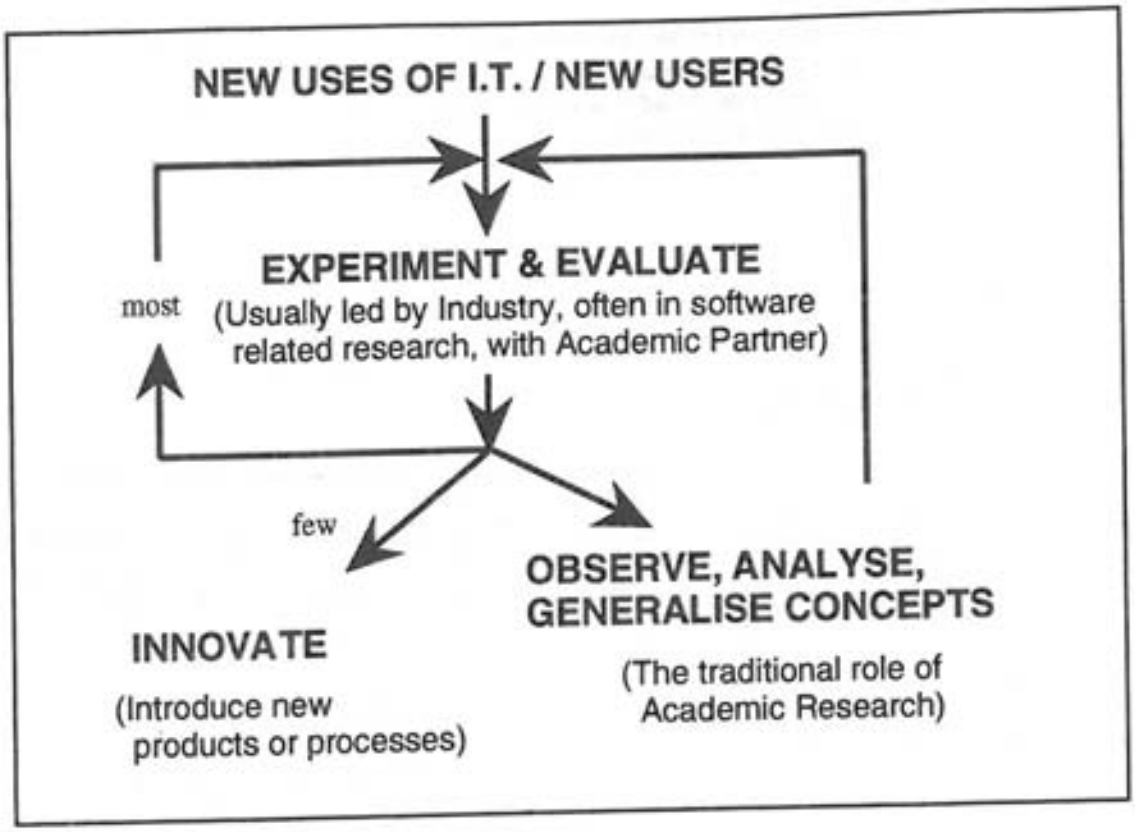

Figure.E.3 Roles in software research.

Directions for Future Research

Though the identification of directions for future research was not the principal aim of the Workshop, it was implicit in many of the Working Groups' discussions and also the subject of a certain amount of direct debate. A number of areas were as a result identified as being amongst those which seemed particularly worthy of additional attention

- Technology for implementing (safely) and managing (securely) large scale heterogeneous systems - with particular emphasis on the problems of scale, naming and mobility, and the integration of legacy systems.
- Technology, methodology and HCI/dynamic user models/adaptive training for intermittent programmers.
- Electronic commerce in general and software distribution in particular.
- Complementary abstraction and specialization mechanisms, aimed at promoting greatly increased software re-use.
- "Frameworks" for new domains e.g. animation, and technology for transferring between domains.
- Human-computer interface, user models, including adaptive behaviour and bilateral communication (co-operative dialogue).
- Work on technology diffusion and packaging to increase use of existing tools and methodologies
- Software technology that goes beyond current object-oriented techniques - autonomy, environmental awareness, emergent properties, intention/desire.

- Artificial intelligence applied to the management of the global infrastructure and the challenges of information access.
- Natural language - a longer term goal but with major payoff.
- Speech recognition and capture, as part of the improvement of the human-computer interface but also as a front end to language translation.

It was also felt by the Workshop, that, outside the computer science area, but with major impact on the way that software will evolve, were two mega-issues. These were:

- Acceptance of change / organizational dynamics: the adoption of IT in organisations - whether society or corporate - is always much slower than early experiments predict. Since the potential for positive change through the global information infrastructure is so high, research on barriers to its uptake is indicated. This research should not be led by computer scientists, but by social scientists (anthropologists were mentioned, not entirely jokingly).
- Intellectual property and value of information: the task here for people knowledgeable about software is to work with lawyers to ensure that precedents set reflect useful and realistic parameters.

Conclusion

The 1968 NATO Conference on Software Engineering was notable for the way in which a set of people from disparate backgrounds - academic, industrial, government - found that they shared a common concern about the then-current state of software development practice. This was thought to be so serious that they used the term "software crisis" to describe the situation.

A notable characteristic of the Software 2000 Workshop was similarly the degree of unanimity among the 22 participants from a wide range of backgrounds. This time the agreement was on the dramatic changes affecting software.

Powerful and cheap processing, storage, networking and software, have created the potential for the global information infrastructure, and changed our definition of what software is, how it is delivered, who writes it and who uses it.

Present levels of hype about "data superhighways" notwithstanding, it was somewhat surprising that there was such a level of agreement, amongst industrialists as well as academics, about the importance of current developments.

One phrase we discussed as the key phrase describing our conclusions was "software joins society". Certainly, the needs of intermittent programmers for usable building blocks numerically swamp those of software professionals. And the pervasiveness of software embedded in devices and to provide information across the net is a major change since 1968.

The Workshop felt that, though predicting the future is difficult - and liable to be wrong - there are signs that the intersection of intermittent programmers with the global information infrastructure could be the significant forcing factor for unlocking new opportunities.

It is likely that we will see massive changes over the next few years, causing the initial PC revolution to pale into comparative insignificance. The Workshop Report contributes to the analysis of the situation, aims to stimulate discussion, and has identified some of the problems and opportunities the new environment will give us.

Chapter 1
Introduction

1.1. Background

1.1.1. The aim of the workshop

The aim of the workshop was to get a small group of people to debate broadly what are likely to be the major changes in the software world by the end of the decade. The initiator of the workshop was ICL, who jointly sponsored it with ESPRIT.

The motivation for the workshop was the belief that a set of discontinuities will be hitting software: technology, techniques, skills and applications, in the next few years. Since there did not appear to be any coherent external body of analysis or thought on these issues, the plan was to get an international set of people together, from industry, academia, suppliers and users to brainstorm the topic.

The workshop was co-chaired by Brian Randell (University of Newcastle upon Tyne, UK) and Bill Wulf (University of Virginia, USA). It took place at Hedsor Park, a country mansion 45 minutes from Heathrow Airport, which is ICL's executive training establishment.

The subject area of the workshop was deliberately broad, and included for instance economics and legal issues - not just technology issues such as software engineering theory and practice..

1.1.2. The participants

All invitations were to named individuals, and participants contributed on a personal basis, and not as official representatives of their companies or institutions The participants were balanced with respect to several criteria, for example type of organization, country, topic area. In order to keep numbers down and have a full set of topics and types of organizations represented, we chose people with broad capability and fulfilling several criteria. We had participants from Europe, US and Japan.

The types of organization that we covered are (in alphabetic order):

- Academic research groups
- Consultancies/Industry watchers representing IT departments
- Consumer and entertainment industry companies
- Desktop computer/workstation companies
- Industrial research laboratories
- Major system purchasers (for example, defence)
- Packaged software producers
- Systems Integrators/Facilities Management companies
- Traditional computer companies

You can find a list of participants, and their CVs, in Appendix A.

1.1.3. Methodology

Having identified, invited, and got acceptance from the participants, we asked them to submit notes of what they considered to be the major factors which are likely to affect Software in the Year 2000, and to write a position paper discussing one of these in more detail. These papers were circulated before the Workshop and provided a basis on which the "home team" (Brian Randell, Gill Ringland and Bill Wulf) could suggest that the areas for work divided naturally, though with overlaps, into three. These were:

i) the environment - economic, social, technical and legal/commercial factors

ii) the business model - the size and shape of businesses incorporating software and the changes (discontinuities) affecting them

iii) the technology of software - how good is it, where is it going, what are the emerging technologies and what are the hurdles to introducing them.

These areas were discussed at a plenary session, and adopted as broad brush headings, though it was clear that there was overlap in several areas.

The Workshop then divided into three Working Groups which each concentrated on one area, presenting initial ideas to the plenary group before refining the ideas and joining in a final plenary session.

Each of the "home team" facilitated one of the Working Groups - Bill Wulf on the environment, Gill Ringland on the business model, and Brian Randell on Technology. They have also subsequently pulled together the excutive summary and taken a view of the report as a whole.

The report is mainly however based on the contents of the final plenary session, the discussions of the earlier plenary sessions, the discussions of the Working Groups and the initial position papers. The main chapters of the report are structured along the lines defined by the respective Working Group. The bulk of these chapters was written by the raporteurs in 48 hours, for which we owe them a great debt of gratitude. Later edits and revisions by participants have been made by circulating this for each participant to comment, and incorporating these into a version frozen at the beginning of May and printed by ESPRIT DGIII - to whom many thanks.

The intended audience for this report is policy makers, educators and industry people with an interest in software - that is, someone who could have contributed to the Workshop, had they attended. We have assumed a knowledge and vocabulary of acronyms as covered by standard IT glossaries (for instance, that produced by the Gartner Group), but have included an expansion of the more specialist acronyms used in the discussions as Appendix D.

1.2. Extrapolations of where we are now

All the participants shared to some extent a view of the current environment and extrapolations from it in discussing the changes seen in software. It was felt useful to capture this level of common understanding.

1.2.1. Economic background

Gill Ringland: The world economic balance is changing, for instance the purchasing power of the newly industrialised countries (S.E.Asia, China and India), in business and consumer markets, represents a business opportunity for industry in Europe and North America as well as an option for offshore manufacturing labour.

The IT business is changing too. With low growth in the G7 countries, the conventional business market will tend to a replacement market, with decreasing price for the same functionality. Meanwhile a spate of joint ventures and partnerships betwen IT companies and those in other industries will explore changes in technology and unlock new applications in markets such as retailing, home education/entertainment.

1.2.2. Organisational factors

Gill Ringland: The role of the large organisation, world-wide, is expected to change; fewer core staff will be employed for a shorter "standard" working life, with most people working for smaller specialist companies, in association with the majors, with lower job security. This pattern has existed for at least the last two decades in Japan (where large organisations have had large numbers of subcontractors who do not operate "lifetime employment") and North America (where large organisations have shown determination and the ability to downsize). In Europe, it goes against cultural attitudes and EEC social legislation. However, mobility of work and leisure, in European business, will increase over the period.

The technology (networks, mobile phones and FAXs, desktop PCs, information services) to support the new "associated" organisation will be important in North America and Japan, and increasingly in Europe.

1.2.3. Factors affecting the business model

Gill Ringland: Many existing IT companies are unlikely to be able to survive the changed business environment.

For instance, business desktops in North America, Japan and Europe are rapidly becoming saturated. Replacement business IT markets will be driven by new applications, and so the nature of the marketing and support effort - and their inter-relation - changes.

In the coming years, the price/performance ratio for microelectronic and enabler products will continue to decrease at the current rate. The consequences are that many more products, especially those in the consumer electronics sector, will have enormously increased functionality implemented by embedded software. Network technology will also leap forward dramatically . But, depending on the regulatory environment, lower costs may not translate into reduced prices.

The shape of the IT industry is changing, with the advent of new competitors - especially the telecommunications, cable, satellite, information, games companies packaging IT services differently, mostly for the home, but also for business. The new competitors (for example, from the entertainment industry) have the cash to run new businesses at a loss, in order to enter new

markets and the traditional IT players may shrink even faster than the market, if newer players have the right skills for the new opportunities.

There will be downward pressures on the price of services. The requirements for customer service will decrease, and so contracts are bid for at low margin costings. Meanwhile, there will be a large number of long-time employees shaken out of major companies looking for freelance or other opportunities to use their IT skills. This will depress prices for professional services..

1.2.4. The industry's technology drivers

One common view was that the IT industry is expected to continue its divergence into customer facing or services organisations and distributor facing or product organisations.

Peter Wharton: So far, the IT industry has been technology driven and although the prime drivers are now the business ones (how to deliver total solutions, the requirements of the end user and so on), technology is still continuing to develop at an almost frightening rate. It is widely expected that the pace of technological innovation will continue and that price/performance curves continue on their current trends into the next century. Mainframe and server prices will particularly be affected.

Tony Temple: This rate of change will continue to accelerate through the 1990s.

1.2.5. The environment for software developers

Tony Temple: We are in a period of accelerating change for developers of computer software. These changes are in all areas of our business:

- Technology for development: The generation of code from specification is becoming more feasible. However, that puts an even greater premium on "nearly right first time" requirements. Technology provides an increased ability to re-use code and built component libraries. The availability of reliable working objects using standard interfaces will allow development projects to grow larger.
- Infrastructure: The system software base is rapidly moving towards a graphical, object orientated user interface, providing standard services to applications. Even though the hardware base will change, these front ends will remain the same, providing transparency. New interfaces will arrive (such as speech and pen) as faster hardware allows. Network technology will improve in performance and decrease in cost. This will mandate a new breed of applications.
- Functionality - new applications: The focus for new applications will include multi-media and network "information miners".
- Commercial implications: Prices will continue to drop. Basic level application componentry will be provided pre-packed with system software at very low prices. More advanced applications will be more expensive, but again prices will continue to fall. Only niche products (by implication with a restricted market place) will be able to command high prices.

Tony Temple: All of this demands a new approach for the traditional software developers. There is considerable opportunity for software development over the next decade, but not all of the existing players will survive these challenges.

1.2.6. The End user interface

Tony Temple: The challenge to the software industry is to make computing accessible to as wide an audience as possible. The bulk of the increased hardware computing capacity will be exploited in the user domain, by more sophisticated end-user interfaces, and by distributing function to individual workstations.

Gill Ringland: The killer IT applications will be those which are attractive to the "technophobe". But the definition of the technically illiterate is changing. For instance, many schools and universities now demand that students can use a PC effectively.

Tony Temple: New technologies (large high definition screens, voice, pen, scanning, and so on) give new opportunities and new challenges for creating an effective end-user environment. Increasing system performance means that a greater percentage of system resources can be spent providing this environment.

Chapter 2
Environment

2.1. Scope of the working group

Public and governmental perceptions of software have an important part to play in shaping the future of the industry. In addition, the global information infrastructure is rapidly increasing in connectivity and bandwidth. As the "net" transcends national and organisational boundaries; new communities of common interest are emerging. Whilst the net's growth raises new issues of management, commercial structure and security, it also brings opportunities for new industries in the area of value-added information services.

The Working Group began by discussing changes in the computing environment since the 1968 NATO Conference. Major areas of discussion were: the public perception of software quality, the potential for the development of the information infrastructure and the benefits which may result. Finally, the group identified some key areas for future research and development.

2.2. Progress since NATO'68

Bill Wulf: For traditional computing systems, problems of the magnitude worried about in 1968 can probably be handled today. We can build a system of the functionality of OS/360 without stumbling too badly.

Cliff Jones: Compiler software is production-line nowadays. In 1968 we knew what we wanted to build, but couldn't. Today we are standing on shifting sands. The environment is changing: that is the difference between now and '68.

Andrew Herbert: In 1968 we had cheap mass storage and were keeping a lot of data on computers, but were overwhelmed by trying to write the many applications to support them. This led to the idea of software engineering for mainframes on which the 1968 conference focused. Now the achievement of connectivity with increasing bandwidth is the technological change that is making us ask what we can do and what the social and governmental issues are. We are now looking at a world with high connectivity where you can essentially have all the media wherever you want.

Bill Wulf: Software obeys something akin to the Peter Principle in that we always want more and consequently are continually working at the limits of our abilities. Research in industry and academia must press on because we are clearly not at a point where we are satisfied with our ability to produce software that is reliable, on time, cost-effective, user-friendly and so on.

2.2.1. New issues

Bill Wulf: There are some issues which are on the "front burner" today which perhaps were not so prominent in 1968. Some of these have to do with a whole-system view, for example that the usability of a system is as important as its functional correctness. We could certainly enumerate examples where the whole system, including the people, failed, not because the software per se failed, but because people incorrectly interpreted what the software was saying. Another issue in this category is that the direction of software development is a responsibility of software engineers, as well as simply producing methodologies and tools to address the

software problems themselves. Software Engineering needs to help everybody, not just specialised software developers.

Chris Phoenix: Many people building information systems do no programming at all.

Bill Wulf: In the issue of software curricula we differentiate between software professionals (the people who build underlying systems) and the professionals in some other domain who use basic software components to solve problems in their area, be it choreography or architecture. Our community also has to take some responsibility for general computer literacy in the wider population.

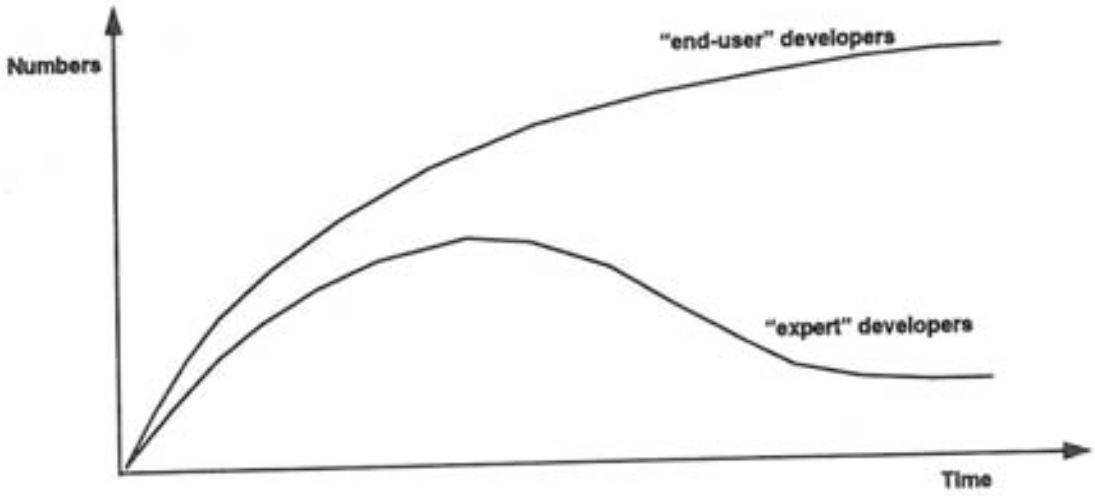

Figure 2.1.

Cliff Jones: As universities, we have a terrible job attracting young people to read for degrees in computing. The real problem is that school teachers do not understand what a degree in computing means. They know what physics and mathematics are, but they do not know which children to point towards degrees in any of the IT disciplines. I would like to provoke people by drawing a graph (Figure 2.1) about the education of IT builders and users. The number of people who are exposed to or need to use IT (upper line) has risen very dramatically from a low start. (The top line includes those who use linear programming and planning packages, but who use software rather than building it.) At the beginning, the number of experts required (lower line). tracked this very closely because you had to be an expert to use the machine. These are now the computer professionals whom we are educating in the universities. My prediction is that their numbers will fall off. The question to my university department is "Whom do we want to teach?". Do we teach the wider group of IT literates, or teach the professionals and face the falling numbers?

David Talbot: What causes the lower curve to drop off?

Cliff Jones: I have no proof of the drop off, - but I believe it could happen, and showing it serves to provoke thought about the possible consequences.

Brian Gladman: The curve (of expert developers) does in fact follow that found in other engineering professions: initially you tend to over-produce, and as the engineering technology matures, you need fewer people, because the knowledge is stable then.

Andrew Herbert: To some extent you are training people to solve the last problems.

Cliff Jones: In some senses I see it as an opportunity, because we might be able to educate some really good system builders.

Bill Wulf: This discussion presumes "business as usual " in the future, but I think we are still in a highly unstable environment. Fundamentally new things coming along will keep up demand for that lower line.

2.3. The public perception of software quality

Brian Gladman: An issue to address is that of cultural predispositions. In mature areas, the understanding of the design, development, production and use phases of products and systems is such that suppliers feel able to underwrite the performance of the products which they market. In contrast, nearly all software comes with a disclaimer and is often delivered with many faults (this is especially true of mass market software where software testing is completed during the delivery of progressive 'enhancement' releases which are really bug fixes). The software industry seems to have a very long way to go here: perhaps we are only a little way up the maturity curve on this basis. I'm not clear whether that's because I, as a customer, am not rigorous enough, or whether there's something inherently difficult going on: whether it's a cultural or technical process that has led to this situation. I suspect it's cultural. It's not technical, because I know people that can deliver software that works very accurately. I suspect we've got used to software being relatively poor, and we've just let it become part of our culture.

Charles Simonyi: This issue is more technical than cultural. The behaviour of software is infinitely more complex than that of hardware. For example, we can write a simulator for most pieces of hardware in the order of hundreds or thousands of lines, whereas our software includes millions of lines, and the complexity goes up non-linearly.

Brian Randell: I think there is a mixture of social and technical. The digital nature of software, the lack of continuity arguments and so on throws so much mathematics and past engineering practice out of the window that it is not surprising people write those sorts of caveat.

Brian Gladman: I don't think there's anything inherent in hardware that makes it uncomplex. What we have done is evolved a set of architectures for handling complexity in hardware that's got it to be manageable. What we haven't got is a set of architectures for software that have actually got us in a position where we can manage the complexity. A chip with ten million transistors on it can be made as complex as a piece of software if so desired. We have matured in the hardware design process to an extent that we do not organise in that way. In some ways, in software we seem to still want to be anarchistic.

David Talbot: Software people tend to shelter behind the complexity argument, irrespective of what they do, and that will increasingly be felt to be an unreasonable situation. I think there's going to be a growing force where reasonable people will say "enough is enough" and will want something better.

Bill Wulf: If the economic incentives are there, it will be done, but it's not clear that society is demanding and willing to pay for quality at very high price.

Robert Worden: I think there's a kind of software Peter Principle: software rises to its level of incompetence. If somebody reined back on ambition, would the market support them?

Les Belady: Software is encoded human thought, encoded for the purpose of execution by an idiot called a mindless machine and this is the underlying reason why it is so unbounded. It is as complex as human thought, limited only by our inability to encode it in such a way that it can be interpreted by a machine. This is not true for any other engineering subject.

Jessica Litman: A prediction: legal regulation will require, inn some amount of time, for software to be warranted. Cars started out without warranties, too. The law stepped in to say "these can do too much damage - you have to back them up". When some sufficiently bad things happen, a couple of jurisdictions will say that software has certain implicit warranties and that those warranties cannot be disclaimed.

Chris Phoenix: The distinction between code and data is becoming very blurred. It can often be the content rather than the program which is at fault. People perceive software to be untrustworthy. How many newspaper articles have you seen on topics in which you are an expert? When you check them for accuracy you often find them deficient.

Brian Gladman: Professional engineers, outside the software community, take extreme views about the low quality of engineering in the software business. In the area of safety-critical systems, there are enormous debates in which some very sound, committed, professional people take the view that under no circumstances should you put software in those domains. However, the error criteria they use are many orders of magnitude beyond the alternatives.

Andrew Herbert: There is general agreement on what constitutes a safe car, but there is no such thing as "a software", and so a general notion of a "quality piece of software" is not the same kind of thing.

Brian Gladman: Embedded software is already implicitly covered by the warranty on the overall product. Warranties on such products usually last for a period of time, and software either works or it doesn't. However, it is built in an "traditional" engineering culture which expects it to work.

Cliff Jones: I have been making the prediction for some time that the lawyers will step in when there has been a disaster, and my fear is that they will step in , in an inappropriate way. The law which will be passed will require something like every line of code will have five minutes' testing. You make a future for lawyers arguing over what constitutes a line of code: I worked for IBM for many years and we had a task force arguing over precisely that. If we do not get advice ready on what would be appropriate laws ...

David Talbot: ... then "the man from head office will come and help us".

Jessica Litman: I expect that the law will require off-the-shelf software to be about as reliable and warranted as any other product. Today we have chips in cars; the cars are guaranteed for two or three years, including the chips. That's going to have to happen to other products, with some adjustments to account for not knowing some of the situations in which end users might apply the software.

Cliff Jones: Reproducibility of faults is an important factor. If I get a bug when I try to do something in Word, I may not be able to reproduce it. Compare this with the detailed thought required to work out the sequence of events which led to the THERAC failure.

Andrew Herbert: In the area of consumer goods, co-operative testing laboratories have sprung up, issuing recognised certificates (for example, the BSI "kite mark"). They conduct tests in response to disasters which have revealed potential faults. We lack similar mechanisms for software because there is no incentive; your product will be later to market than the competition. At the moment, people are getting away with letting the market do the testing for them. Part of the perception of poor software quality also comes from early mistakes: misconceived systems such as the TAURUS UK Stock Exchange disaster should be halted before they are executed.

Jessica Litman: The public do not only see software failing in off-the-shelf products, but also in commercial systems. Hospital over-billing is an example.

Brian Gladman: The public then assumes that safety-critical software is of the same quality.

Jessica Litman: My guess is that today's younger generation will be more critical of the technology, because of their increasing computer-literacy.

Brian Gladman: It seems to be impossible to insure software components in the UK today. Although there is software in insured products

Cliff Jones: For instance, oil-rigs.

Brian Gladman: This has not, to my knowledge, been tested in the courts.

Bill Wulf: I do not want to deny the importance of producing quality software, but the issue will be resolved, not by breast-beating, but by legal and commercial forces: banks and insurance companies have not been willing to pay for security, for example. At some point it will become an economic issue but, until then, not a lot can be done.

Brian Gladman: Market differentiators have to be used by some companies, and it may be that there will be a paradigm shift caused, not by customers like us, but by some suppliers recognising that quality is a marketable commodity.

Bill Wulf: This may only appeal to the more sophisticated customer. A better business model for the larger market is the one where I, the software supplier, will pay you $10 for every bug you find. The consumer is interested more in output than quality-ensuring processes like formal methods.

Jessica Litman: People are cynical about advertising based on the advertiser's process. If presented with good support services, they will not be afraid to buy a PC from, say, Gateway.

2.4. The new information infrastructure

2.4.1. Industries Converge

Bill Wulf: A great many forces are acting on the software industry: for example, intellectual property concerns, an evolving merger between computing and entertainment, and so on. Among the greatest of these forces, I think, is an enormously expanded and significantly changed telecommunications web - the "national information infrastructure" as it is called in the US. The existence of this infrastructure will facilitate, if not demand, information services that we are all probably too myopic to see now.

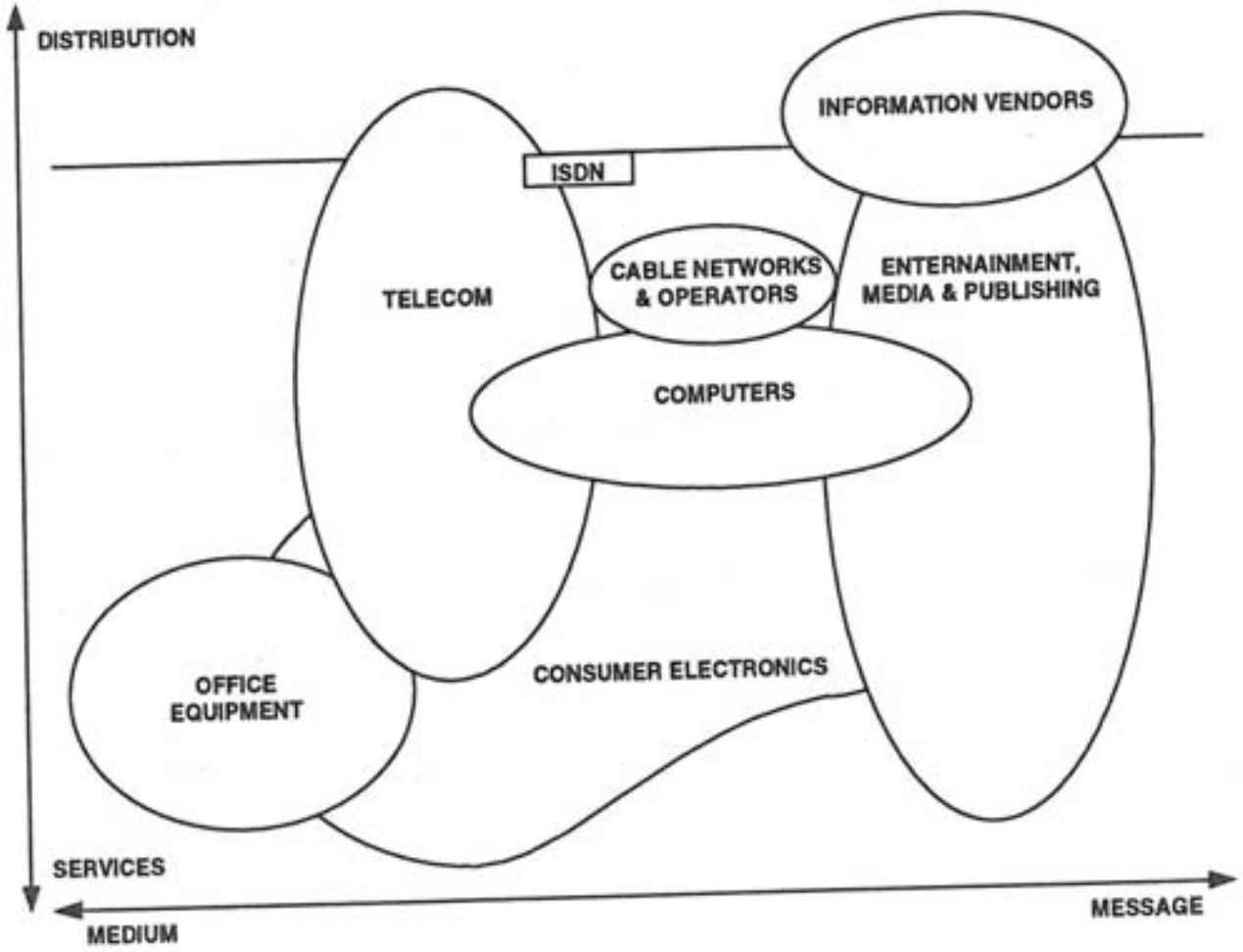

Figure 2.2. Industries converge.

Andrew Herbert: The information industry, the entertainment industry and the embedded software domain all use communications and are starting to converge. We are on the threshold of total connectivity for all media. This represents a major discontinuity. Figure 2.2 - based on concepts from the Harvard Business School - shows the relationship of the IT industry to industries sharing technology and customers.

A lot of information is being pumped into the ether: the information infrastructure is not only accessing stuff in databases, there is an opportunity for live manipulation of data about things that are happening.

Bill Wulf: Examples are:

- the US Intelligent Highway project.
- devices that tell you the current average speed on the motor ways around London.
- the Berlin traffic project.
- the Channel tunnel.

Cliff Jones: Considering what can be done with the information infrastructure, I certainly expect to see a much bigger difference in the next decade than I have seen in the past one, and maybe even in the past twenty years, as a result of real information flow across the network.

Bill Wulf: We need to be clear that the information infrastructure is not just the wire and fibres. The information is what is important.

Brian Gladman: The nature of applications of computing in society is changing, because of this ability to handle information on a global scale. The integration of wide-area communications and computer systems has given society a new capability, new challenges and new ways of doing business. Internet is an early (even amateurish) demonstrator of some of he potential for change.

2.4.2. New classes of application emerge

Brian Gladman: We were talking earlier about where some of the new applications will come from. There are one or two areas where I think the information infrastructure, and in particular the global information infrastructure, will spawn a massive range of new killer applications. I was one of the early members of the Internet and this has now become enormous. What we are beginning to see is a new class of application coming on line. People see viruses as a threat, but we are researching them as productive applications that roam the network collecting information. (We are already using them as open source mechanisms and producing very big savings - for instance getting information on US legislation takes 10 minutes rather than the days by other means. We don't keep information standards inside CISE (my part of MoD) any longer, because we can just go and get them off the network). Tasks such as updating directories will be done by roving programmes that roam the networks and build them. One name that has been coined is "knowbots".

The fact that we have this global information infrastructure is changing our way of doing business.

Andrew Herbert: Most computers spend most of their time doing nothing, so we need to ask the question "what could we have them do that's productive?". Wandering round the world, finding things would be ideal, so long as it's cheap enough.

Brian Gladman: That has actually been a very productive use of the Internet. MoD is watching with interest projects that are taking resource off the Internet. Just by launching a distributed application into that environment, making use of the spare time on all the machines, you are creating a massive computing resource; far bigger than we have in the MoD! One particular experiment, called RSA129, to factor a 129 digit number, was set as a challenge several years ago and expected to require next generation computers to complete within a reasonable timespan. The answer, after 3 months "on the net" is thought to be within days of completion. This sort of application is being done in an amateur fashion at the moment, but you can see that amateur work being translated into more commercial applications.

Andrew Herbert: Your machine has the potential to turn into a jackdaw, which goes around collecting interesting things for you, without you being connected all the time it is "active". There are all those machines sitting there that could be doing productive things, sifting through the information that's out there, talking to each other, collecting things, structuring things. Will that happen and what will it look like?

Bill Wulf: I don't know how many of you have seen this data, but the annualised rate of growth of the world-wide web is 300,000%.

Jessica Litman: There's quite a growth of organisations like America On-line and CompuServe. What's happening in the US is: families are signing up, because the kids are going to college and it is a cheaper and easier way to keep in daily touch than the telephone. The children, meanwhile, are getting net connections from their colleges, and saying to their parents "there's this really neat Usenet News group"; or "you can get a recipe through CompuServe".

Andrew Herbert: One of the ways world-wide web is interesting, is that not only can you access the information, but it is very easy to contribute to it. If you do some analysis or research, it is trivial to add it to what exists on the web and link it back to the sources. That is one of the reasons it has grown so fast; not only can you get things from it, but you can contribute back to it, which is really turning people on.

Andrew Herbert: I can get connectivity by lots of routes: for example if I'm an information provider or an information consumer, I would certainly buy my connectivity from the phone company because it's probably there anyway. But who do I go to do the broking of the information services? Now the telecomms and cable companies want to claim that piece of territory. In countries where the telecommunications agencies are government agencies, they will take advantage of their monopolies and their governmental influence, to try and maintain that piece of territory. In less regulated worlds, you are seeing all sorts of entrepreneurial organisations springing up and I think we need to understand what those organisations might be; will they hang off the side of the paper publishing industry, the entertainment industry or the computer industry; will they be small garages full of bright kids, who think there is an opportunity here.

2.4.2.1. Structuring and filtering information

Cliff Jones: I worked for many years for IBM and in the mid 1970s I remember vividly a discussion with one of that company's senior executives. Basically he was talking about a vision of the future in which companies might provide "computation grids". The analogy was with the National Power Grid which provides electricity to our homes and the idea was that we would be able to plug into this grid in order to do computation. I saw this as completely irrelevant to the future of IT. What I wanted to see was an information grid rather than a computation grid; something which would enable the ordinary person to access and possibly manipulate information which was important in order to make their lives either more productive or more enjoyable. The reaction from the executive was amusing - that project would be too expensive for a company like ours!

We're in a situation where there is an enormous mass of data available in machine readable form. We are now moving to a situation where we can begin to access some of this data over World Wide Web, Gopher, and so on. But this data is not information, it is just not structured in a way which makes it useable. Furthermore there is very little opportunity to manipulate or connect other pieces of information into the data which is accessed as ASCII files. I am convinced that information technology for "real people" is about access to information (not computation per se - but computation will be required in order to manipulate the information into the form that the user desires).

Information is structured data, not strings of bytes. (I don't want to go into details here of what formats are or are not desirable but something like a collection of entities and relations seems to me a reasonable working hypothesis.) To a first approximation there is zero useful information in this sense available to me at my workstation today. If I list the tasks I wish to undertake, either in my professional work or to support my outside interests, I find that the tools available to me reduce to things like 'grep'ping a large series of locally stored ASCII files and limited access to locally designed databases. What I want to do is to access data stored and maintained on other machines and to be able to connect information of my own into those databases so that when I next look at it I can be reminded of my previous interaction with that data.

Again, let me report an anecdote - this time a recent one. We were recently approached by an engineering firm who reported that they needed help with database technology. During the meeting that we set up in order to understand their problem, it became clear that their task was indeed access to information. The firm is involved in designing water treatment and sewerage plants. They need information about population and trends in the change of population; they need information about the terrain in which the potential storage and processing facilities could be built; they need information about the weather and the way in which rainfall correlates with geographic information about rivers and their rates of flow; they need information about the state of pollution in those rivers in order to plan their processing plants. The good news is that much of this information is available in computer databases; the challenge they face is to access these totally separately designed databases and correlate all of the information into a coherent picture. In fact I formed the impression that these water processing engineers were actually turning themselves more into database interface experts than physical engineers. There is for them a huge economic advantage to be able to access and process information rapidly.

Andrew Herbert: You're going to need someone who does the customisation and filtering. It happens already. My eldest was doing some homework and wanted to know something about the Duke of Wellington and I said, "let's throw it at Mosaic" and see what it finds. It came back quite quickly, with the minutes of the city council of Wellington, New Zealand rather than information about the Duke.

Cliff Jones: That is a fundamental structured information problem. I believe identifying Wellington, the person, as what you really want information about, to be a key technical problem that will actually hold up this superhighway.

Andrew Herbert: If your model is that there are brokers in the middle, I would know that if I wanted to know about the Duke of Wellington, I would go to a historical research broker, as opposed to a geographical broker.

Cliff Jones: Is that broker a human being or a program? I believe the technical challenge is how to "architect" this thing so it can be a program.

Brian Gladman: Certainly, how we build clients that have got semantic intelligence is going to be crucial, because there is so much information out there.

Andrew Herbert: I agree, it should be a program. The next question is, is it a program running in every domestic interactive TV, is it a program being run as a service by someone on a piece of infrastructure, and you then have the meta-issue of how to find which broker to use.

Cliff Jones: Chemists are now highly computerised, but they have got hundreds of databases; they need a broker just to know which database to go and consult.

Bill Wulf: Historians have a 250 page volume, which is nothing but a list of the names of sources of data.

Cliff Jones: We need to structure information in order to make it accessible. I don't like this idea that you just have to go off and hunt in unstructured databases

Brian Gladman: I think it is a mixture. There are two views of the world. One, you structure the information first, and then you go and develop around that structure. The Internet view is, you get out there and do it and then you evolve structure from that unstructured process.

Bill Wulf: There is a class of historians who spend most of their lives in archives, very flat information spaces, with little structure imposed on them. They would argue, very vehemently, that any preconceived notion of what the structure is will lead you down the wrong paths. Even any preconceived notions of what the mechanisms are by which you would impose that structure.

Jessica Litman: Fifteen years ago, the law market started with two databases. One was Lexis which allowed full text searching, but added no organisation or indexing. The other was Westlaw and it was all organisation, you could not do full text searching on Westlaw. Westlaw hired lawyers who read cases and coded them all. You could retrieve stuff by code or by topic, but not by text. It was much faster doing it that way. After 5 years, Westlaw moved to full text searching. The other facilities are still there, but I don't know anyone who uses them.

Brian Gladman: Information is a connection between raw data and some overall purpose. Until you state the purpose, you cannot actually derive the information. Therefore, that semantic connection can only be made in that relationship; it can't exist in the raw data. The risk is that any interpreters will put a distorted slant on the information; you do need access to the raw data.

We came across this in our information systems. For instance we put information systems on ships and we were reducing their performance, because we hadn't noticed that information is only inherent once you connect that purpose to the underlying material.

The client actually has to be intelligent; it has to establish a semantic connection between the purpose that the client is serving and this raw information. There ought to be a basic grammar which allows you to describe your semantics to a standard searcher. I need to have a much more intelligent way of describing my search strategy.

Alessandro Giacalone: This is essentially the area of knowledge representation. It is a well established community that aims to supply tools to represent a domain of knowledge.

Cliff Jones: But I don't think raw text is enough. For example, information used for historical research, such as that in the public records office in London, represents a challenge because of its poor organisation. If you could get that information, first of all on machines, secondly in machine readable form, and thirdly, with some structure, it would significantly cut down the search time.

Jessica Litman: There is definitely a market for people who will structure information. It would be a pity however if you could only get your information with added structure.

Brian Gladman: It is often the case that secondary records don't truly represent what is in the primary records. You can have the secondary records, but you have got to have the primary records as well.

Cliff Jones: One of the reasons I argue against flat text is that it is trying to reproduce what we do with books. I would like to see computers give us much more structured information. In fact, maybe this is the end of the book, because what is the point in storing things in books in the way that they are already conveniently bound and distributed through libraries and book shops. What we want is access to information at any depth.

Andrew Herbert: There is a straight marketing point here. If one can provide on-line information, the more I structure it so that there are pointers which entice you to stay logged on and follow them, the more of my stuff you'll browse. If I just give you flat books, you'll read one of them and then turn it off.

2.4.2.2. Many disciplines are affected

Bill Wulf: Over the last four years I have traversed a personal mini-odyssey that I think is highly relevant to the future use of this infrastructure, but in ways that I do not yet see clearly. In particular, four years ago, with typical technological arrogance, I presumed that all humanistic scholars were technophobes, unable and often unwilling to learn to turn on their word-processor. I was wrong, and I now believe that:

- information technology will have a larger impact on scholarship in the humanities in the next two decades than on the sciences, and
- the uses of information technology in humanistic scholarship is a better model of what "everyman" wants, and we should pay a great deal of attention to how to support their uses.

Andrew Herbert: You are then dependent on making the people who understand how to structure information, like historians, geographers and librarians, able to use IT, to put in place the linkages. They are trained to make the necessary judgements and we need to give them the tools by which they can capture those judgements and build the information structures.

Andrew Herbert: Another factor here - given we now have more forms of monitoring and more computers and more forms of presentation, is that we've actually got a lot of different sources of information which can be cross-correlated. The net can provide a lot of information on-line about the same kind of thing, so you can cross-check.

Jessica Litman: That's okay so long as the information is independent; so long as it's not all derived from the same source.

Chris Phoenix: There are a lot of people building systems who are engineering information services. I am concerned that there is no specific discipline behind this.

Brian Gladman: When building these new kinds of systems, we have no effective measures for information. There is a whole series of theories of information; Shannon theory is an example, but there is a need for a lot more research.

David Talbot: The emphasis is shifting from software engineering to information engineering.

2.4.3. New industries will emerge

David Talbot: A consequence of the infrastructure is the new opportunities for business and jobs. What you have is a whole mess of data at the bottom, you then have people that want information. The whole business of information services, from capturing information in digital form, through to say, helping water engineers to link geographic information with weather forecasts, represents the new economy this infrastructure can create.

Chris Phoenix: The barrier to entry to providing information services on the Internet is very low. A whole new industry layer is emerging - this is shown in Figure 2.3

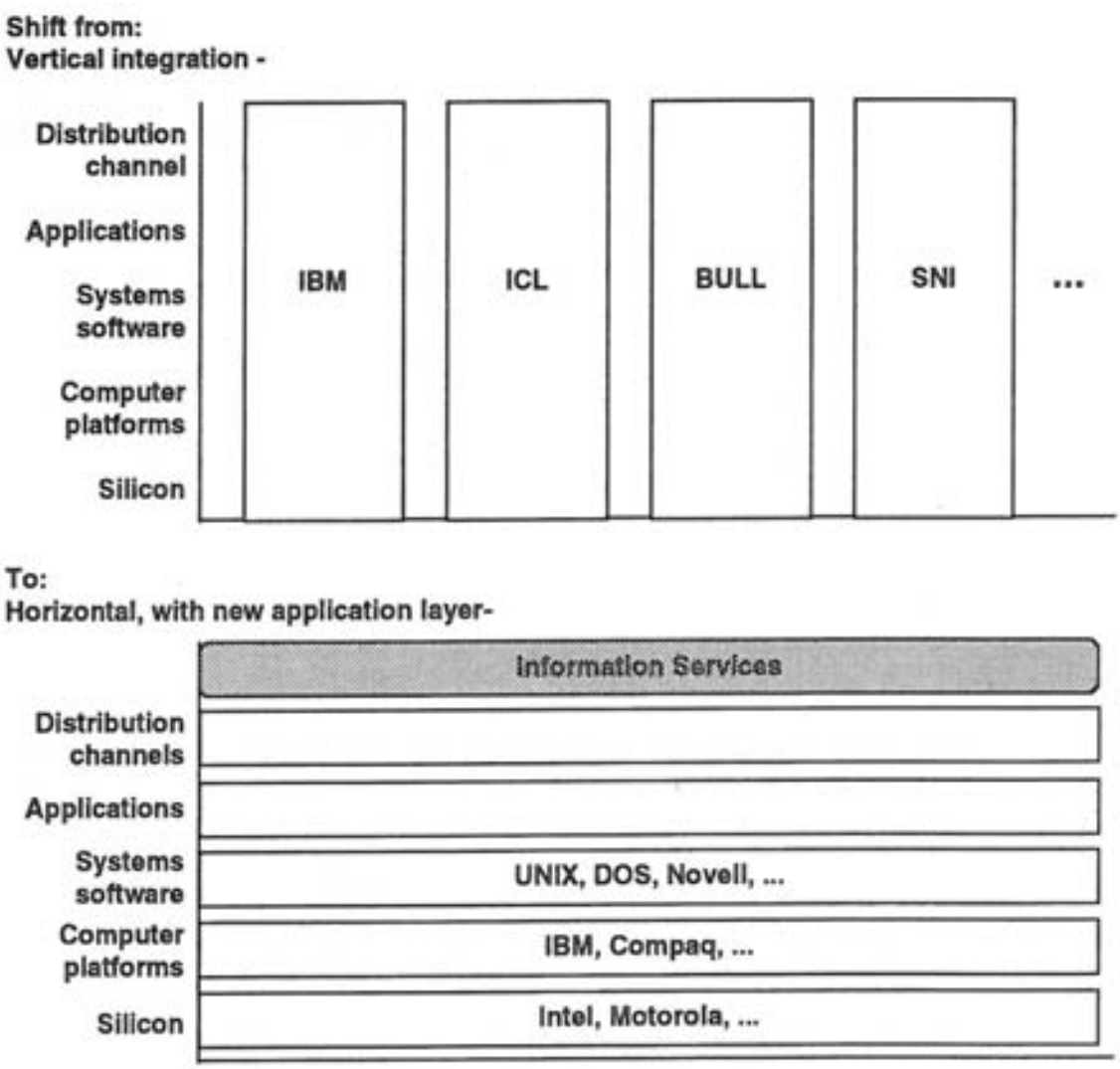

Figure 2.3. Shift from horizontal to vertical integration

Andrew Herbert: Even small players can get to a wide market.

David Talbot: It provides for people who struggle with current channels of distribution and communication with the outside world. It has already built several opportunities. In the current circumstances, an information distributor would be local, in a corner shop. If he has information that's of general interest, he's now got a ready made channel of distribution. It's the users who will produce some of the most original applications.

2.4.3.1. The role of government in enabling the new industries

David Talbot: We still don't have good access to information. We only just have the connectivity. There isn't much digital property available (that is, information in digital form).

Bill Wulf: Whereas industry can perceive the value of adding value to raw data, you may need government to do the job of digitising information sources.

Brian Gladman: In the formation of a new industry, the initial processes have to be driven by government. It will not come from industry. That early phase is too big for any individual company. For example, no company would digitise the information in the British Library. That has got to be done by government.

I do an awful lot of research in particular areas, which is actually very slow while you're dealing with books, for a fairly specialised subject, very few libraries have the material, so librarians have to do multiple references. I can ask for an article, which can take three or four weeks; it often ends up coming from the US. If the information was electronic, it might only take minutes, which would revolutionise the way I work. It's old material, not new things, that is the problem.

If you want to start up a new industry, the government has to provide a very strong incentive for that process, using its weight to get that industry off the ground. Our investment, as government, should be in that start-up phase.

Bill Wulf: The human genome project, with its huge amount of on-line data, will spin-off an enormous number of industries.

David Talbot: Government launches infrastructure. Almost nobody launches infrastructure privately. The issue is, what could be classed as infrastructure; that is infrastructure that requires government initiative as opposed to private initiative? Does it also constitute some of the digital resource, as well as the highway itself.

Brian Gladman: The communications infrastructure is there to a large extent, but that is not true at the information level.

2.4.4. Social and legal consequences of the expanding information infrastructure

Brian Gladman: The nature of the application of computing in society is changing, following the integration of wide-area communications and computer systems. At one level, we will see fundamentally new applications arising from ability to handle information on the global scale. Geographic distribution is the key issue. History has told us that discussion of ideas reduces the risk of political conflict. The global infrastructure is already having an effect in this area.

There are enormous potential social opportunities in a more integrated world where national boundaries play less of a part in people's thinking. Some would see these opportunities as a threat. It is possible to establish instant communities, not just in scientific, technical and academic areas, but social as well.

Andrew Herbert: With the rise of telecommuting on the network, you can choose to live where you want to live, as opposed to living where you want to work. I think this will definitely have an effect on demographics.

Bill Wulf: My community or job is not where I live, it's those with whom I interact.

Jessica Litman: As an academic on sabbatical it's commonly my experience that people don't know I've gone

Andrew Herbert: We are saying two things: that the expansion of the information infrastructure may rebuild physical communities because people will not be charging around in cars all the time; and it also enables virtual communities for work and play.

2.4.4.1. Electronic commerce: bringing order to the Net

Jessica Litman: Electronic commerce requires security. People buy goods by credit card over the telephone - merchants don't even ask for your name any more: they just want your number. If you try to buy anything over the net with a credit card, any reasonable supplier will say "This is not a secure medium. Call me on the telephone.".

Bill Wulf: I believe that both because of small transaction value and security, I would prefer a centralised billing service.

David Talbot: I think centralised billing will be a service along with a number of other services.

Brian Gladman: As network systems become a part of our business activity, it will be vital to protect their integrity and that of the information which they provide and this will require improvements in the security which such open systems can offer. Software will play a vital role in this, since a number of vital algorithm-based techniques for providing security services in globally distributed information systems are now emerging.

Questions:

- is there an acceptable mid-course between those groups who wish to see this technology widely deployed and those who wish to restrict it?
- given the international nature of the problem, where should decisions be taken on the balance between wider deployment and continuing constraints?

Andrew Herbert: We are getting an information infrastructure that is making people aware that you can find things electronically; electronic commerce is the next step from that. With electronic commerce, the information infrastructure becomes a subsidiary issue. Electronic commerce is the end result, the infrastructure is the means of getting there. The infrastructure will cause people to want to start doing transactions, based on the information they have access to. That will lead them to do electronic commerce. And then we will talk about how we will "transactionalise", so to speak, the information infrastructure.

Bill Wulf: Right now, the existence of the infrastructure will drive the development of commerce.

Andrew Herbert: That's when we might see the shift from anarchy to organisation. While it's essentially free and co-operative, anarchy is fine. As soon as you start bringing the commercial things in, there are issues of trust and protection, and that's where you'll start to add requirements. But I think you have got to get quite a long way down this path, before people will want to start dealing with the commercial activities. So I think you start anarchic, and then parts of the anarchy are rather more constrained. They're the parts we are prepared to start doing trading in.

Jessica Litman: When credit cards started, there were a whole lot of things built in to reassure users. You got your charge slips back for example, so you could verify them. Those little reassuring features have fallen away over time, as people's level of comfort with electronic money has increased.

Andrew Herbert: All that paper was there, because the credit card was trying to mimic the old system, and people needed to have confidence in the new one.

David Talbot: The old system became the new system, there wasn't in that sense an "old" system.

Andrew Herbert: I think we might find that we'll have these anarchic structures. Some will try and operate commercially in the way in which we do electronic commerce today, but we'll also find that the commercial process will evolve, to exploit this new world. What we end up with, in terms of our support for trust, could well be quite different to the current trust mechanisms. It is hard to predict how these will look.

The challenge is the transformation from the informal information infrastructure to the commercial information infrastructure

David Talbot: And there is a continuing cycle, as more and more information gets pulled into the infrastructure..

Brian Gladman: We expect domains of order to evolve within this broadly disorderly structure.

Bill Wulf: And we would advocate broadly following the Internet evolutionary model.

Andrew Herbert: We could envisage a transition from highway to marketplace.

Chris Phoenix: There are some people who would think you meant motor way service stations!

Andrew Herbert: To summarise, the order in which things happen is: the academic anarchic world is building the information network, which is beginning to be used for commerce. IPR is based on notions of fair competition. You can't claim too much IPR if it would disadvantage other people, but you protect yourself to some extent; it's all a question of judgement. If the network starts off in this rather anarchic, freewheeling society, the IPR threshold will be set quite low, because so much is going to be contributed openly and publicly. Where people start trying to claim territorial rights, they're starting from quite a difficult position.

Brian Gladman: The Internet comes from the academic community, with its ethos that all information is shared. We need to retain those ethics.

2.4.4.2. Education: Internet in the classroom

Robert Worden: Will every application have good built-in training/education/help? Will multimedia I/O (voice, pen, video) be built into most applications? Will applications converge on a few standard "personalities" which most users know and can operate, or will there be constant pressure for innovative interfaces? Will a younger generation, computer-literate from childhood, develop a fluency in IT which leaves our generation open-mouthed and obsolete?

Bill Wulf: Although we hear much about the installation of computers in classrooms, IT has had a very limited impact on higher education. What is the problem? I think all we have done so far is replace drill, perhaps making it a little more flashy and interactive, but it is still drill. The exciting aspect of the application of IT in the humanities is that scholars are now using computers and communications to do research, and that can be moved into the classroom almost instantly. This leads to a teaching of the process of scholarship, of discovery, rather than teaching the products of scholarship. The quality of difference is enormous.

Chris Phoenix: Current projects encourage co-operative working between schools across the network. This will be a major driver for the development of the information infrastructure. It will also dramatically change the economics of education delivery.

2.4.4.3. Social computing: re-enabling the forum

Andrew Herbert: Computers are valuable for running simulation models. Many educational activities are associated with participation in a simulation. The games market is moving away from conventional games to games based on participation in simulations. Coupling a city simulation game with information about the real world allows the public to participate in planning decisions, for example. Democracy was originally about collective argument, but the communities have become too large, so we elect politicians. If you have more widespread access to information, modelling experimentation, you can make an impact on the political system.

Jessica Litman: SimHealth, made widely available at cost, enables people to debate proposed US health reforms. But the MAXIS company had to build in certain assumptions about what will cause the system to break and what will cause it to flourish. My guess is that people will be as distrusting of simulations that violate their intuitions as they are of reports in newspapers of the opposite political persuasion to their own.

2.4.4.4. The standards process

Robert Worden: Will broad standardisation moves converge, or will there always be important new areas where standards have not converged?

Andrew Herbert: In some respects standards are helping - for example, all databases now talk SQL. In other respects standards aren't helping - the various bodies standardising UNIX seem mostly to have added further variants to the species rather than reducing them. The computer industry is losing interest in the ISO process as being too slow and divorced from business needs. Industry groups such as OSF and OMG have made rapid progress, but seem to run out of steam once they've reached first base, or else they become tainted as vendor camps in competition with one another. Users have little confidence in standards, except where major industry sectors can gang up and form purchasing cartels, such as POSC.

How is all of this going to develop? Will we see more user cartels? Will we see vendor alliances replacing the de facto and de jure standards process? Will we see a small number of "killer products", licensed by everyone, providing the base technology?

> The group agreed unanimously that the traditional formal standards process does not have a hope of working any more.

Bill Wulf: I think that's very important. Everything that's been important in this industry in the last two decades, as far as I can tell, has been a de facto standard before anyone got around to making it formal, if then.

Andrew Herbert: There are important things done through the processes engendered by the standards mechanisms and there are different kinds of standards processes. Standards activities can build a sense of community direction out of which technology emerges. That clearing house is important, whether the standard exists or not, as a quality check on the work of its participants. Another kind of standardisation involves a subset of the industry which is about to converge on a similar kind of product: it is worth getting together to see if you can get that convergence. Sometimes it proves impossible to perform the revision necessary to take the standard to a second version.

The standards processes are ways to achieve public confidence by providing fora. They each have strengths and weaknesses. The international process is good for reference models and scoping areas, but it is a hopeless way of agreeing technical standards for particular jobs. The industry standards process is quite good at pinning down instantaneous pieces of technology. These should be coupled together and used for their strengths.

The Internet did not have a set of standards, but has a set of practical processes by which standards could evolve, through the task forces. They provide the fora for discussion, improving quality and ensuring acceptable quality for the users. We still need that process, but we probably do not do it through ISO.

Good technology is extensible. Because the Internet designers knew that people would be experimenting, they thought about making all the fields and data structures extensible. They didn't get it all right, but they have let a lot more things happen than many other pieces of "standardisation" have managed.

Brian Gladman: Standards in the global information structure are not technical (for example, authentication standards). They have to be developed with an understanding of the environment in which they fit - on a multi-cultural basis with lawyers, technical people and even politicians.

2.4.4.5. Management of information systems

Bill Wulf: There was recently a failure of our campus PC mail services. As a consequence all overnight mail was lost. That generated a major hue-and-cry. The argument flashing across the bulletin boards was that this would have been intolerable in the telephone system. The e-mail system is more important to us than the telephone system. That was one of the first times I have seen the general campus community up in arms about an issue like this - not just us techies.

Andrew Herbert: Management is not a widely-explored issue in computing. Even in the telecomms world, the management is at the level of the modems and switches. In terms of managing the system to achieve a purpose, it is not touched. There is no doubt that it is a black art.

Bill Wulf: How many organisations have network control centres?

Chris Phoenix: Quite a few are just that: control centres. They do not run a service management function. Customers tend to want the system management in the product, especially if you offer integrated solutions, as we tend to at ICL.

David Talbot: This is part of the drive that encourages people to out-source. There are many people managing systems. The issue is the confidence of the basis on which they do this: it is like programming squared. Managers are often told to get on with it with no prior experience: "one management task is much like another, isn't it?".

Brian Gladman: Once a relationship is contractual, with money changing hands, you are forced to think through the requirement better. In-house you may be dealing with resources you cannot control.

Cliff Jones: I would not like us to encourage out-sourcing. If a company which relies on information out-sources its whole computing strategy implementation, it runs the risk of putting its information supply in the hands of a third party, who may be providing information to their competitors as well; they don't know what their business is any more.

Chris Phoenix: A company, or any organisation, needs to own the problem of managing its own information resources. That does not mean it cannot operate an out-sourcing policy on its IT systems.

Andrew Herbert: You have this paradox of wanting total flexibility, but a more managed system. You can't resolve the paradox, so you have to haggle, which leads to out-sourcing. You may get it wrong, but you can get it right. An example is where London Underground IT department got rid of a whole batch of straightforward DP tasks such as running the pay-roll. That freed up funding for a research department who built a geographical map of London from ten feet above the ground to a hundred feet below. This was a useful thing for London Underground's excavations, and they could sell a service based on it. They therefore moved from an IT department which was costing them money to one which was making them money.

2.4.4.6. Charging Models

Andrew Herbert: Will software be purchased over the network? How will accounting be done, especially between information providers - for example, who pays who what if I put your cookery recipes in my on-line "lifestyle" magazine. I guess this strays into copyright issues, but also begs questions about policing, charging, licensing and so forth.

Tony Temple: Distribution and charging for software have become commercial issues. Network distribution of software is possible now, but how can it be charged for and policed? Should we be investigating payment by usage?

Robert Worden: There will be a thriving market in software components: Solutions to the problems of advertising, distributing and paying for software components over the net will have evolved. High-value components will be leased on a pay-per-use basis. Standards for interfacing utility components (OLE2 and so on) will not converge, because requirements keep changing.

Brian Gladman: I anticipate that a wide variety of information services will develop on the Internet, offering a range of charging models: for example, subscriptions for large users and per-time basis for smaller users.

Andrew Herbert: Compare this with billing models for telephone services: domestic subscribers have itemised bills; large institutional users negotiate a fee for an annual capacity.

Bill Wulf: In the early stages, billing will be messy, but we will possibly end up with a centralised billing service.

David Talbot: I understand that some 34% of AT&T's revenue is spent on billing, 17% of BT. Surely one would have expected that they would have sorted these out and made economies of scale, but no. I would expect us to develop a very menu-driven system of centralised billing.

2.4.4.7. The infrastructure provided by the entertainment industry

Jessica Litman: People talk about the expansion of the information infrastructure as though the basic advantage of it were instant access to 500 movies. More interesting, and what will drive the expansion, is not the longer menu, but the fact that new applications will allow connections between people: not entertainment or education as we know it, but collaborative rather than interactive applications.

Andrew Herbert: The current information infrastructure is used primarily by professionals, who use it because it comes into their offices or laboratories. Most of us who are enthusiasts start on it that way: we have made the effort to get it into our homes by modems and the like. The entertainment industry, because it is geared to mass distribution, is getting high-bandwidth multimedia connectivity into homes. I think these are going to merge, with the same infrastructure being used to watch 500 movies and log on to the Internet.

In addition, we have a more computer-tolerant generation growing up, for whom electronics are a given, and who have the right cynical attitude about computing systems.

I know that there is a risk involved in this form of expansion, that it will be dictated by the needs of the mass-distribution entertainment industry, but the important thing is to get the screens, keyboard and mice into people's homes.

Brian Gladman: If we are sensible, we will now try to avoid those things in the engineering that prevent the applications we would like to see. A lot of TV distribution is being designed in such a way that it could not support the other applications ...

Jessica Litman: And the people who own current TV financial interests are concerned to wire our homes to keep that so. If we are planning on piggy-backing on their electronic infrastructure, it is worth nothing that their model may be the wrong one. They may offer us either old movies or "interactive" entertainment that allows you to interact with the entertainment. I'm not sure that either of those is a saleable concept.

Brian Gladman: Political and business factors could drive the infrastructure in a direction that is not neutral in respect of the information that can be delivered, and market pressure can drive to the lowest common denominator.

Bill Wulf: The entertainment industry's model of intellectual property may not be fixed. For example, the notion of derivative works is, in this network environment, going to go through the roof. There will be hundreds or thousands of layers of derivation and will we really all have to pay royalties all the way down through that list? ...

Jessica Litman: The problem is more serious than the problem of paying all those royalties. The entertainment industry's model at the moment requires you to get the copyright owner's permission before you even touch their material. (Under that model of the derivative works right, if you watch a movie tonight, and then you go home and think about the movie's ending differently, you had better have asked the film company first). In the USA now, the entertainment industry people are saying that they will keep their vehicles off the information highway until there is some coercive method of protecting their intellectual property. I believe the entertainment industry is bluffing. I think we should resist requests for leak-proof enforcement methods. Once such mechanisms are in place, we will not be able to get rid of them.

The social challenge down the line for the software industry is whether, if the entertainment industry insists on playing by its rules, the software industry will be able to say "Fine, but we want to use the same hardware and play by our different rules, that make sense to us, rather than music rules or movie rules."

Brian Gladman: We need an infrastructure that does not predicate particular IPR models.

Cliff Jones: The entertainment industry might provide us with charging models which actually work. We might find it hard to do Internet charging (small amounts by credit card) separate from this.

2.4.4.8. Jurisdiction

Brian Gladman: I think the issue of jurisdiction in the global information infrastructure is unclear. The UK does not have a freedom of information act, but the US does. It is therefore possible to obtain information about UK government activities through the US. There is now no clear boundary between the nations in the infrastructure, so the concepts of jurisdiction can become difficult.

If the global information highway is to work, we must find international mechanisms to set the rules in place. Some countries deny the existence of patents. There will be a migration of certain things to those countries on the electronic highway because they won't have the restrictions, in the same way as shipping is registered in Panama.

Jessica Litman: Jurisdiction is based on place, on physical presence. The point of the infrastructure is that physical geography is no longer important. Until there are treaties to resolve the issue, people will be claiming jurisdiction over anything that has an effect in their territories.

The rules between countries are not as different as they could be. We have had diverse computer decisions in the US, but recently, courts in the US have been coming to the same conclusions about software copyright as courts in the UK. The network is global, and the fact that there are no boundaries makes it easier. One of the reasons why we have not previously been able to do anything about something happening in, say, Thailand, is that it was within Thai jurisdiction. Once it hits digital media, though, it isn't inside Thailand any more, it is all over the world. As soon as it hits the borders of the UK or US, the rules that the world has set up are in action.

Brian Gladman: I cannot see how these rules are to be established. I can see a set of national rules, but I can provide myriad examples from the Internet where those rules are ignored.

Jessica Litman: Absolutely. Copyright rules are like sodomy laws: they are ignored routinely by everybody.

One level of complication is that what motivates governments in helping the information highway to come into existence is the notion that it will give some nations an economic or jobs advantage. Governments have not yet realised that there are no meaningful borders or boundaries: you cannot appropriate a piece of the net and say "this belongs to us". Thus, in the US, one of the political arguments that you hear too often is "We have to have these rules because that will help us create more jobs and take them away from our trading partners in Europe, Asia and Latin America.".

Brian Gladman: Do you feel it is sufficient to run the Internet with just our sets of national rules?

Jessica Litman: We have more sameness among countries, and there is greater agreement internationally, on what the intellectual property rules should be than in almost any other area. The intellectual property rules (patent and copyright) are more alike country-to-country than the liability rules.

Brian Gladman: Trust relationships between supplier and receiver in different countries seem to be particularly difficult. I think some of these issues will not be finally resolved until some bad things happen. A good model from the UK is Citizen's Band radio, banned by the government, but legalised after the population imported radios en masse. The Internet style is right: governments will have to recognise that those constraints are meaningless.

2.4.4.9. Policing the Net

Brian Gladman: Global information system networks and the connected hosts are vulnerable to being attacked by introducing viruses, worms and so on, or by attacking the availability of the communications services on which they depend. The Internet is vulnerable in this respect since it is wide open to abuse. In the main it has relied heavily on its users behaving responsibly and, considering the size of the network, this has worked remarkably well with only a few known major attacks on the network as a whole.

Andrew Herbert: This says something about public response. If you are told something is bomb-proof and defensive against you, there is a certain urge to try and fault that system. If you are told that the system depends on your co-operation to keep it alive, you approach it in a different way.

Jessica Litman: You never get 100% leak-proofness. It's the sense of community that makes the Internet self-policing.

Andrew Herbert: The dependency in Internet on unwritten rules and common convention leads to a need to educate accountancy, management and political professions to understand the role and limits of electronic auditing. We should encourage detection mechanisms and ways of supporting sensible behaviour.

Jessica Litman: In the US, one argument put forward is that inadequate policing will lead some suppliers to refuse to put information on the super-highway. I don't believe that. Television took off without any intellectual property protection at all. Most countries just allocated frequencies. Because the threat of withholding information from the super-highway is coupled with the promise of jobs, we may move prematurely to a system of electronic auditing in a way that will distort the growth of the medium.

2.4.4.10. Ownership of Information

Jessica Litman: Have we been overprotective of intellectual property rights in software and other computer-related works? Is intellectual property law hindering rather than promoting innovation? What aspects of software works should the law leave unprotected?

Do we need new legal models to extend intellectual property protection, or to enforce intellectual property rights, in an era in which distribution of new works is more likely to occur through electronic transmissions than through the purchase or lease of tangible copies of protected works?

Jessica Litman: What is going to be privately owned, what will not be susceptible to private ownership, and what will be a hybrid? We come from societies with a tradition that information and ideas are free: while they have value, no-one owns them. You can add value and charge for that service, but you cannot appropriate the information.

Legally, a good analogy is to water rights laws. There are different ways of regulating rights to use water. One is an appropriation model. In the Western states of the US, where there isn't much water, when you appropriate it, you own it. You may not have created it, but by virtue of using it every year to irrigate your farm, you have established property rights in that water (or, more precisely, to that volume of water). In the Eastern States of the US, you have the right to have water continue to flow through your property. You don't have any business appropriating that water so that it no longer flows through the property downstream. You can do stuff with the water, but you can't capture it and say "This is my water. I have a certain number of acre-feet.".

Information is like the water.

The first question, then, is: can we or should we allow people to appropriate or own property rights in information? If we do, it provides an incentive to make the information available to everyone else, but it also erects entry barriers against future "downstream" users of that information who might want to use it or make it available. Right now, people's views on which

we should do tend to be more religious than they are empirical because we do not have the economic data to figure out which model would work better.

A central axiom of intellectual property law, in every country, is that the public retains the public domain to be mined by anyone, but intellectual property laws give people enough of an incentive to engage in mining, to, get their products together and to distribute them. What we do not know in the digital age is how much is enough? Some would like to err on the side of giving out lots of property rights so that everyone will turn their attention to appropriating and distributing. Others point out that lots of industries grew up under an umbrella that protected people from intellectual property liability: the juke box industry in the US happened because someone figured out that there was an exception written for something completely different in the copyright statute, that allowed people not to pay ASCAP royalties; the cable industry has overtaken broadcast television in part because the cable industry had the benefit of being exempt from some of the copyright laws affecting others; the phonograph industry - disks, cylinders and piano rolls - initially grew up because of an exemption from normal copyright laws. One set of views asserts that if you want an industry to take off, you give it some breathing room. You do not burden it with a bunch of "homesteads" and tell it you can't do anything without getting permission.

Bill Wulf: A lot of case law seems to based on tenuous analogy, and does not go back to the questions of how to promote the public good, but rather refers back to an arcane case. How do we get the discussion focused on first principles?

Jessica Litman: You can't. Cases take place in the context of a dispute where two people pay their lawyers to make the best arguments. The most successful arguments in courts have been the ones where you draw an analogy with something familiar to the judge. For example, one might argue that a program is just like a play. If the plot of a play is protected by copyright, then the structure, sequence and organisation of a program is protected by copyright. Not only has this been employed successfully, but judges to whom the argument has been made have given copyright a much broader scope than judges to whom other arguments were made. So of course the lawyer who is trying to achieve a result in a given situation will make that argument in the courts.

The political process will not solve this problem. When the US Congress is faced with the need for an amendment to copyright laws, it delegates this to the companies who have strong financial interests in copyright. Our DAT (digital audio tape) bill is as long as the rest of the copyright law itself because many different interests had to sign to it off, and kept making additions.

I think software people need to be better consumers of legal services. When lawyers say "look, here are the rules", companies should argue back.

Brian Gladman: I think we, as technical people, should plant the right analogies.

Jessica Litman: It would help enormously if we could all make our intellectual properties laws simpler so that lawyers did not have to be there to explain them to techies.

2.4.4.11. Liability

Andrew Herbert: There is no practical way of imposing liability on information brokers if they are, as we expect, small organisations. Large organisations are more worried about this, because of the importance of the image they have built up.

Jessica Litman: Liability tends to follow the availability of insurance coverage. If insurance can be sold, liability will be imposed. If insurance cannot be sold, few suits will be brought.

Bill Wulf: The clear implication is that we should not improve software quality because, if we did, the insurance companies might begin to insure, which would therefore lead to liability and the consequent collapse of the software industry!

Jessica Litman: Software companies' parents are insured. At some point, liability is going to be imposed. Right now the rules are confused. In the US, we have 50 different jurisdictions and rules. Some suits have been brought against individuals for malpractice. This follows the services liability model: did you fall below the standard of care that is routine in your field at your level of expertise? The rules by which that model is applied vary a lot locally. Certification of software engineers certainly makes it easier for liability to follow, but need not drive the decision. In a professional service situation, you don't need a certified software engineer to be able to tease out the standards you would apply, it just makes it easier.

There is a product liability model which is based on foreseeability. Liability for off-the-shelf software or interactions is dependent on reasonable anticipation of the uses to which the software might be put. The liability rules for products with embedded software are the same as the rules for those without.

Brian Gladman: Perhaps the quality of products in the information systems domain is driven by the liability issue: a challenge made against the quality of such software might drive some improvement in the product.

Chris Phoenix: Again, I draw the distinction between software and the information it manipulates. I am much more concerned about information pollution than software pollution. If, for example, I, as a professional lawyer, use a legal database to extract a structure and set of interpretations, which I then publish across the net, and changed it by accident or design, is that malpractice?

Jessica Litman: That is one area in which US rules differ from the rules in other places. We do have laws about negligent misrepresentation of information, but it is more difficult to recover, because of the US investment in freedom of speech, the first amendment and the like. These rules are in flux, as the information economy replaces the commodities economy.

2.4.5. The Vulnerability of society to IT

Bill Wulf: There are clearly some areas in which we have big exposures: not only buggy software, but human factors issues such as those which contributed to the USS Vincennes disaster. I suspect that, in those cases where disasters are precipitated not by a computer, but by its improper use, society will not give us the benefit of the doubt. If it has to do with a computer, it's our fault.

Andrew Herbert: How many computer disasters have actually made the front pages of the nations' press. Most of the discussion of the Airbus A320 being fly-by-wire has been in the technical press or technically-literate television and radio.

Cliff Jones: People are woefully ignorant of what the exposures are, and we need to educate the general public. Recent reports in Newsgroups indicated that over 50% of the 4,000 Sizewell B (a nuclear power station) software tests failed, but manufacturers say this is due to the test rig being wrong in nearly all of the cases. I'm not sure what confidence I derive from the tests it did pass. That should be on the front page of every newspaper.

Alessandro Giacalone: The issue of correctness or quality depends on the area of application. A lawyer was using a tax program with a design bug which caused it to give wrong advice in certain situations. On the other hand, doctors accept the output of ultrasound foetal monitoring machines and sometimes they infer the wrong diagnosis, yet some will still not accept digital output of X-ray images and image enhancement. It seems that the idea of correctness therefore depends very much on the expectations of the application.

Brian Gladman: I am concerned about how to find a balance between freedom of access to the information infrastructure and its protection from abuse. In the development of these infrastructures, people have gained knowledge of systems that have put them in sufficiently powerful positions to be highly corrosive if they chose to be so.

Bill Wulf: We will find a balance between individual and social rights. Can we collectively have a right to protect ourselves?

David Talbot: Is there not a "right to use" issue here as well, a question of what would constitute an acceptable level of universal access? We have a right to post, from wherever we live, an item of mail and expect it be delivered in the same condition for the same price whether we live remotely or locally.

Brian Gladman: And how are such rights to be established globally?

Andrew Herbert: In some cases we have installed barriers to access. For example, there is a considerable but not widely-available literature on bomb-making. Do we prohibit information going onto the network, or do we police who is accessing it?

Bill Wulf: Because this involves issues of deciding what is right and wrong, we cannot answer the questions, but it is vital to stimulate a public debate.

Brian Gladman: One major concern affecting the growth of the information infrastructure is the attitude we take to whether we should expose any discovered vulnerabilities to those professional people who can do something about it, or whether we should keep this information secret.

Jessica Litman: There is a risk that the Internet may propagate "urban myths" such as the Neiman Marcus cookie story. A lot of effort can be wasted debating these untrue stories.

Andrew Herbert: We need to have means for tracking the worth, value and origin of information, so that you can differentiate between source and derived fact, and you can trace the derivations to sources.

Jessica Litman: That has to be one role of the academic community: to keep the data stream honest.

2.5. Directions for future research

Bill Wulf: There is definitely a continued need for R&D in software engineering. For example, formal methods aren't widely used, in part because the research problems haven't been solved.

Cliff Jones: Particularly in concurrency.

Bill Wulf: And there is the whole issue of automatic verification.

Brian Gladman: I would be interested in Cliff's view of how much the technology really has permeated into industrial application. My feeling is that there are many applications where formal techniques could be applied and they just aren't. People are still ignorant of the area.

Cliff Jones: Yes, but that's true of any technology transfer. It doesn't seem to be taking abnormally long.

Andrew Herbert: A lot of the automation is missing. Civil engineers use a lot of mathematics, but it comes in software packages. They key in the parameters of their bridge and the program runs away and tells them "yes, that's a bridge" or "no it's not".

The mathematics and the concepts have been around for long enough, and maybe the maths is more practical, but people who continually write programs see formal methods as another thing to learn - and it is not well automated. If they could somehow exhibit their programs to something and have an interactive dialogue, they'd be more tolerant of it.

Cliff Jones: There is a definite need for theorem provers. That's a piece of technology that needs cross-industry funding. No one company will be prepared to put money into building industrial strength theorem provers. You can argue that we'd like to be a lot further on in applying formal methods, and you can say: do we want theorem provers, but I don't think the take-up is much lower than you would expect.

Brian Gladman: In the Health Service there have been radical differences in terms of performance in applying these advanced techniques. Some companies are really right out there in the forefront, others don't even know what they mean.

Bill Wulf: There is an issue of the cost benefit of these techniques. We have to make them more cost-effective for simpler things. You won't get formal methods applied to Word 7.0, because of this.

David Talbot: Introducing process change is always extremely difficult.

Cliff Jones: We are also trying to adopt new process in an area where people have no mathematical training.

David Talbot: All these ideas represent huge levels of opportunity in creating new businesses, new jobs and new levels of wealth, with the potential for more international co-operation. However, this is all predicated on solving some major issues. These issues are not simply issues of implementation; they require substantial levels of R&D to be applied. There are real possibilities which need to be unlocked - now.

Chapter 3
Business

3.1. Scope of the working group

The main topics the working group focused on were those that were changing in a rapid fashion - who produces code (what is programming?), what are the forces for change in the market, and in what areas did the group see research (or innovation) needed.

The work of the group was, in most areas, oriented towards scoping, rather than detailing, since all the participants are aware of the difficulty of forecasting the future outside areas tightly coupled to technological hardware advances.

The area in which we saw the major changes was in the extended number of people who will, over the next decade, "program" in the sense of using sets of structured commands creating sequences which may deliver "right" or "wrong" answers depending on the skill of the programmer. Analogies were made with the skill of car driving, which had a similar introduction path but now is done - with varying degrees of skill - by up to 400 million people world-wide.

3.2. Why will the business model be different?

3.2.1. Defining 'business model'

Les Belady: To start, can I ask a clarification question? What is meant by 'the business model', and what is 'business model discontinuity'?

Mike Braude: Let me take a shot at that. It actually encompasses a wide range of things, such as:

- the products you sell,
- who are your target buyers?
- do you make or buy various parts of your offering?
- are you going to be a low cost supplier, to build market share, or will you be content with smaller market share?
- what distribution channel do you establish...

Les Belady: So a vector of properties of the business, different descriptors of how a specific business operates.

Mike Braude: Yes. You can think of it as the things you must tell a venture capitalist to get money.

Gill Ringland: And 'discontinuity' refers to a rapid change in factors that influence the success of a business model. For example, the recent collapse of hardware prices has resulted in much lower margins. This opens questions on availability of funding for future hardware research.

Bill Wulf: With a discontinuity in the assumptions that underlie the business models, profitability can be changed dramatically and many people can become unemployed.

Les Belady: (winking) Well you see I thought that the shift from continuous to discontinuous in economics was similar to the shift from analogue to digital in electronics. (laughter)

Gill Ringland: The sort of trends that form the environment are the areas in Figure 3.1. These form part of the picture, though they are not specific to software. What possible changes or events that would result in discontinuities? Which more specifically affect us in software?

Figure 3.1. Trends.

3.2.2. Effect of software on the IT industry

Mike Braude: The biggy is the fall in software prices ever since unbundling and especially with the switch into desktops away from mainframes.

Gill Ringland: I've been seeing a set of changes in the business models of the IT industry which are knock-ons from this price erosion in software, for instance in services. Do we know where is the money is being made in the various products and services related to software?

Mike Lesk: There is lots of money in content, there is money in operations, there is no money in the software, and absolutely no money in the hardware. If you look at things like DIALOG, the on-line commercial databases, fifty percent of their revenues goes to the content provider, 10% for telecom service for the connection, and 40% for the service.

Mike Lesk: The breakdown of revenues [in the DIALOG service] is 50% to the information supplier, 40% to the network operators, 10% to the telecoms suppliers.

Les Belady: Do those numbers make you happy Mike?

Mike Lesk: It doesn't make me happy but it does open up the market, so that there are lower barriers to entry for niche information suppliers; this ought to be an interesting place for entrepreneurs. They can make their profit from the information, using software components from the industry to mechanise access.

Gill Ringland: The effect on systems integration - project based delivery of systems - has also already been significant.

Tony Temple: Improved development methods will allow either user support staff or brought-in consultants to build custom vertical applications to meet business needs both cheaply and quickly. It seems likely that such integrators will build run-time environments with little or no licence charges - all the price will be up-front.

Robert Worden: For routine "housekeeping" applications the trend towards packaged solutions is inexorable; outsourcing is another manifestation of the commoditisation of this sector. Only for those applications which define a company's competitive edge is there a case for building unique bespoke solutions, and even these are built more and more from standard building blocks - hardware and software. However, the building blocks do not easily fit together.

Robert Worden: We see the major problems in this as capturing the requirements fast and delivering the system in a shorter timescale than they change (laughter).

Mike Lesk: Aren't you finding that customers are looking for decreased customized content as a way of containing this risk?

Robert Worden: Yes, though of course we've also specialized in the projects which are at the leading edge and we believe the difficulties which beset these projects now (lack of application definition, fluctuating and conflicting requirements, communication and co-ordination problems) will be just as prevalent in 10 year's time.

Mike Lesk: Some of the US majors have had to lay people off because of the trend we are seeing to increased use of components and the effect of that on project price.

Gill Ringland: The lower cost of software components is also a problem in the software component industry itself.

Tony Temple: A few companies are now emerging who specialise in developing or buying software objects, and who will sell the resultant object libraries to integrators.

Prices for software components are dropping, and will continue to drop. Eventually basic level application components will be provided almost exclusively pre-packaged with system software at very low prices. More advanced applications will be available at a higher price, but even here competitive pressures will continue to drive prices down. Only specialised niche products (with a restricted marketplace) will be able to command high prices.

Distribution and charging for software have become commercial issues. Network distribution of software is possible now, but how can it be charged for and policed? Should we be investigating payment by usage?

Gilles Kahn: It is hurting us at INRIA as we try to set up companies to exploit software developed internally, the cost of marketing is often too high to be worth it.

Mike Lesk: We've been through this with some software we had - the cost of packaging it up for sales and supporting it was outside what we wanted to do as an organisation.

Mike Lesk: The overall effect of software unbundling in the software industry has clearly been to drive prices down - what is the position in the embedded software industry which is still bundled?

Remi Bourgonjon: We have been looking at a number of applications - for instance HDTV - where the software becomes effectively unbundled, but I don't see a "new software version" for a shaver being on the horizon.

Mike Lesk: I also see a number of changes in software pricing which affect the model again downstream. One is changes in pricing mechanisms - simultaneous use licences vs. per copy charges. Shareware isn't catching on.

Les Belady: Maybe because of piracy?

Mike Lesk: Certainly new technology is needed to manage software assets and billing, which would reduce piracy, and might open up a price difference.

3.2.3. Changes orientated towards software

Chris Phoenix: The newer model has technologists much closer to the user than the traditional model.

Robert Worden: The users are operating without an IT department you mean?

Chris Phoenix: Yes, this means a different support structure from the software suppliers, help desks and on-line enquiry, a whole new industry.

Mike Braude: Software pricing is a very big issue in the 1990s. The pace of software innovation has slowed down compared to the boom that followed the introduction of the PC. We expect a gradual erosion of pricing and this will reduce R&D expenditures. Thus there is a positive feedback loop, resulting in slower innovation and growth, unless there is a major improvement in software development productivity.

Mike Lesk: Here is a possible change. Software coding becomes unimportant, information supply becomes more important, that is, the world moves to table-driven everything. By which I mean that, in the world of the future, people don't write as much new code, they feed new data to old programs. And the set of standard utilities becomes very flexible and able to deal with more kinds of information. But the writing of new code becomes a relatively less important part of the business.

Robert Worden: I've got a counter to that. Aren't you saying that we've moved the virtual machine up one level and that tables are the new game?

Mike Lesk: That is another way to look at it, yes. That people write spreadsheets rather than programs to compute economic models because the spreadsheets are powerful enough.

Remi Bourgonjon: I have some doubts about that. That's probably true about existing areas, but what about new areas where you will still have to write a lot of code.

Mike Lesk: Agreed, but what I'm trying to say is, that in a new area like making movies, it is going to turn out that the amount of work to write programs to do speech synthesis is going to be small, in comparison to the number of people that do not generate code.

Mike Braude: I have two questions. First, do you think this applies to commercial enterprises like banks, insurance companies and retail chains? Do you think they will employ fewer programmers in the year 2000? And second, what is your evidence, because I don't see any indication today and I would expect to see some indicators of a changing trend today. The germ should be here.

Mike Lesk: My argument is, basically, looking at GUI software and the rise of GUI toolkits as a replacement of people sitting writing *x* programs one after another.

Mike Braude: But that is equivalent to the old prediction that people will just write specifications and use code generators as opposed to writing code itself.

Mike Lesk: That's right.

Remi Bourgonjon: But what about the many new areas, for example control systems? Home control and lighting control. There will be a half-megabyte of code in a vacuum cleaner to control its sensors. And this will require a lot of code writing.

Mike Braude: But Mike Lesk's statement is that everyone will do 'programming' like everyone can sing. And we do it at different levels of proficiency. And many will just be filling in tables, and not writing any procedural code.

Les Belady: I believe there will be two kinds of software activity (elaboration in submitted paper). In one, we can write clear specifications and these will be the 'pure software programmers'. But the far vaster work force will not be primarily software people but will be involved in software.

Gill Ringland: Here we touch on the environmental issues, we can see that the liability of the two groups could be different. If you produce software your are responsible for the software performing as advertised. If you build bridges, using software models, you could have liability for whether the bridge falls down. If this turns out to be because of faulty software, you could in turn sue the application supplier.

3.2.4. Time frame and potential for change

Mike Braude: I'd like to make a point. The trouble I have with Mike Lesk's assertion is, if it is true, it confines companies to certain business models that are comprehended now and that is not acceptable for a lot of businesses who need to add value. This is the reason that packaged applications have not taken off in the commercial world - because a packaged application imposes a certain business model of how to do that business. If you adopt one, you are limited by some external person's view of the business.

At least in the business world I think this is unlikely. You can't overthrow the old business models and do something revolutionary if you use the old canned software that other people had. Maybe it could happen in the operating systems world.

Brian Gladman: We are seeing the emergence of component based systems construction and a supporting industry infrastructure (that is a component supply industry). A fully mature discipline will have several industry supply 'layers' and there will be very good, long standing standards, especially simple ones (5v logic, the standard house brick etc.). In the software world the position here is not clear - some standards have emerged (languages, library interfaces, application portability interfaces etc.) but it is not clear that there is a true industry infrastructure which supports component based construction. Some company specialisation is now occurring and there may be the signs of emerging component supply companies with the development of C++ but it is very tentative. On this basis I suspect that we are low on the maturity curve.

Robert Worden: But maybe there will be pre-built modules that allow businesses to put together systems quickly and capture added values in the unique combinations.

Mike Braude: Maybe someday, but the year 2000 is getting very close and the pieces aren't there yet. Most of those guys are still running COBOL on 3270s. I just don't think it is a realistic projection for existing businesses in this time frame.

Remi Bourgonjon: Existing areas don't change that rapidly, you're right. But there are all kinds of new areas coming which will have new paradigms.

Mike Braude: But the trouble with that (and it is the reason that code generators haven't penetrated business) is that each of these businesses is very different from the others and the procedural code needed to run them is unique. And the problem has not been solved, and it may not be solvable, how to figure out general structures that you can take down to the detailed procedural code that you need to write to run a specific insurance company.

Robert Worden: I think there is a nice dichotomy there. Those two viewpoints are both plausible possible outcomes, alternate views, and I think we should present both of them.

Mike Braude: In fact that would be a good paradigm for how we do all of our stuff. In order to do justice to one line of reasoning, you need to see another and the dialectic lets you judge between them.

Brian Gladman: Another point is that mature industries exhibit a specialisation of roles and a clear set of relationships in carrying design, development and production forward. Thus in the construction industry there are architects, designers, surveyors and construction companies with a well established set of relationships in carrying a task forward. Such specialisation is less evident in software engineering.

3.3. Estimating the magnitude of changes

3.3.1. Estimating who produces code

Gill Ringland: We should try to come up with some kind of measures that indicate the types of changes we have been talking about. Can we get some quantitative measure of who produces code?

Mike Lesk: Bellcore maintains 50 million lines of codes. That is a real number and I think it is too big. I'm not certain if knowing that helps.

Remi Bourgonjon: One measure of the rate of change is that the amount of code in consumer products is doubling every year. I'm talking here about 'C' level code. The growth rate is exactly the rate that the price of hardware is going down.

Les Belady: Let's consider the group we might call 'Software vendors'. This includes Microsoft, Oracle, the companies that make software to sell to others. How much of the code written each year is done by these companies in contrast to all the other software written?

In the second group we have the embedded software which is not sold as software but is included in some other product or service. This group also includes all the IS shops in companies that produce software for internal consumption. Let's just make up some numbers to try help imagine how the situation might change.

Robert Worden: Let's estimate what the ratio is now and what it might be in ten years.

Les Belady: Yes, good. So let's guess. Current ratio could be 1:10 and the future ratio could be 1:100.

Mike Braude: I think 1:10 is too low, it might be 1:100 now. That is, the software companies in the strictest definition of the term, the software vendors, created about 1% of the new code in 1993.

Robert Worden: That seems a little too much. Maybe 1:20 or 1:30 would be better.

Mike Braude: So can we compromise on 1:50? I'll take that so we can move forward. I don't think the absolute number is important, what we really are interested in is the relative change. I'd say the ratio is changing 15% per year.

3.3.2. Projections of software developer populations

Mike Braude: The world-wide hardware industry is about 150 billion US dollars (10 to the power 9) and software is about 35 billion (ISV only).

Mike Lesk: I heard a few years ago that, in the US, for each dollar spent on hardware there were two dollars spent on software. That included all software costs such as internal IT support staff, and so on.

Robert Worden: Using our 50:1 ratio here suggests the world-wide software market is $1.75 trillion (10 to the power 12) US dollars.

Mike Lesk: If we call it two trillion and we assume programmers cost $100,000 per year, this implies 20 million programmers in the world. Is this too high?

Les Belady: Remember that is the total industry. If we back out the 50:1, there are 400,000 software engineers employed in the narrowly defined 'software vendor' group of companies.

Robert Worden: That provides some check. We created an internally consistent set of numbers to describe the situation. Perhaps after the discussion we could get on the Net and ask if anyone has better numbers available.

Gill Ringland: What is the trend? How does the 20 million change?

Mike Braude: In ten years it will probably include half of the world-wide white-collar work force. So what is that? (calculates) 200 million people? What kind of growth rate is that?

Les Belady: If we choose the year to be 2010 (16 years in the future) that is about a 17% compound annual growth rate. And a model that may suggest what we are thinking is mathematics. There are very few professional mathematicians in the world, yet each of us uses mathematics everyday and usually with out thinking 'now I'm doing math'. In the future, professional people in most fields will use programming as a tool, but they won't call themselves programmers or think of themselves as spending their time programming. They will think they are doing architecture, or traffic planning or film making. (Figure 3.2 below presents the group's estimates of who programs.)

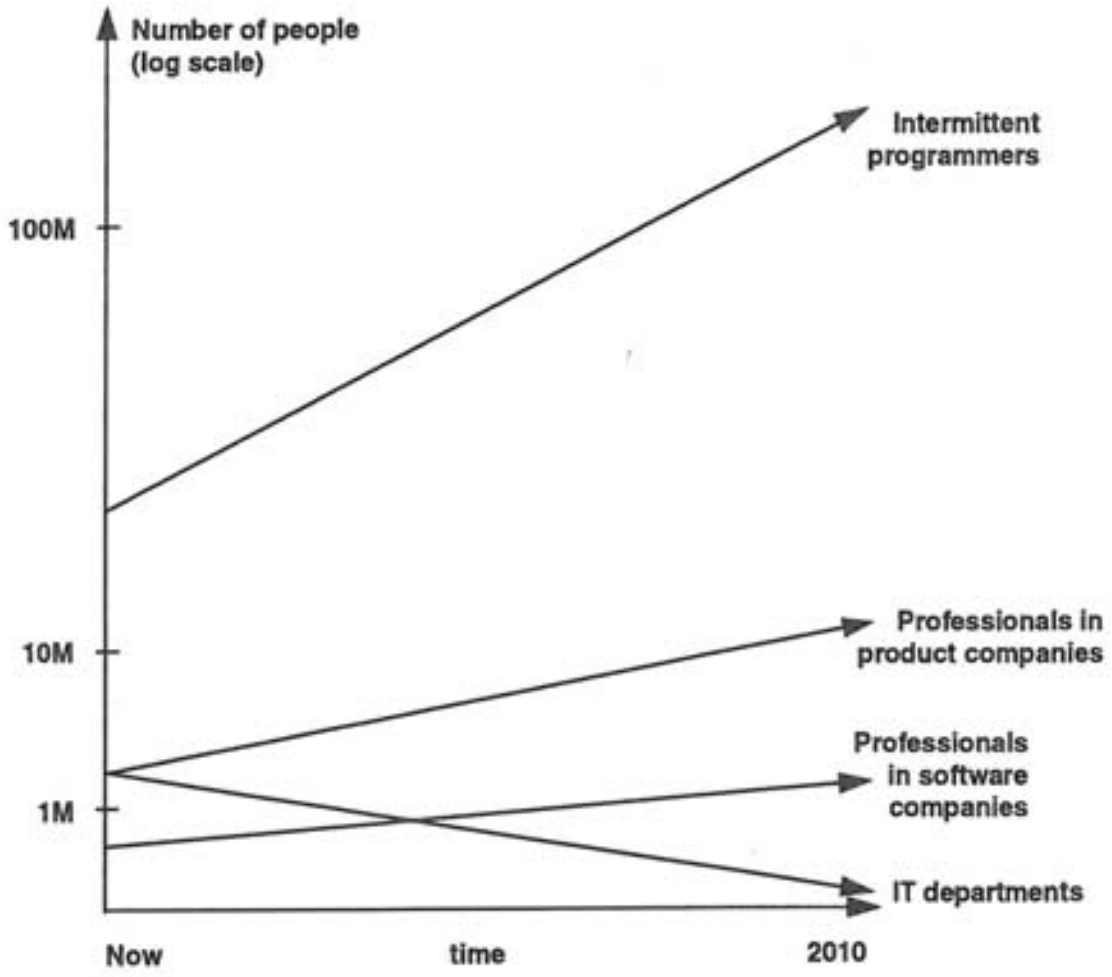

Figure 3.2. Who programs when?

- "Professionals in software companies", refers to those programmers who work for software vendor companies such as Microsoft or Logica.
- "Professionals in product companies", produce code that is embedded in some product, it is sold to a customer but not as software. Example companies include AT&T, Philips and Sega.
- "IT departments" are programmers whose code is used by their employer for various internal functions.
- "Intermittent programmers", or perhaps "End-user programmers" work on some function (architecture, accounting) where they program as a tool to accomplish their job but they are the main users of the programs they produce.

Les Belady: There was some debate about the inclusion of numbers in the report because these are only guesses made by the experts in the business group and there was concern that numeric values would imply greater precision than was intended. It was decided that my name would be included as the source because I will be the first to retire!

Mike Lesk: Reading is a good historical analogy. Centuries ago, kings did not have to know how to read; they employed a scribe. Today we expect the president to read as part of his job, he doesn't think that he was elected to read.

Gill Ringland: Driving and chauffeurs is another example. I think we are confirming a comment I heard earlier today, that in the future people will use software as they use electricity today. But let me ask, can we place these people geographically? For example, will many of them be in India?

Mike Lesk: One of the questions that comes up in the United States is that hardware costs a lot to ship around the world but we lost much of the manufacturing to the Far East, software costs nothing to ship but we have lost little of it to the Far East.

Les Belady: Ah! There needs to be a classification into well specified problems which can be shipped anywhere (you ship specifications) and poorly specified problems that require the software engineer to be near the problem, and thus learn from it while coding. Now a question to you, could you delegate to India your razor software? The production of the software?

Remi Bourgonjon: Actually I was in India a couple of months ago to find that out. You are totally right that the problem is specification of requirements and the stability of the requirements that are evolving. And the problem is user interfaces which are poorly specified.

3.4. Forces for change

3.4.1. Where are the business drivers for new software

Bill Wulf: I see two major categories that will drive the changes in numbers of programmers. The first is more powerful, more usable, less expensive tools. This will impact the professionals and end-user programmers. The second force will be the general permeation of computers and software throughout society. The growing use of computer games and entertainment, personal organisers and cellular phones, will educate people and raise their expectations about how to do things. As we saw with video tape and the Minitel, pornography will be an early heavy use. I expect to see celebrities used in advertising software products and services, especially for things like games (like they are used to sell running shoes).

Mike Lesk: Entertainment will be a major force in the software business. It will produce a demand for better graphics, it will product economies of scale and standards for new features. And it will probably push the business toward having more money in content and less in pure technology.

Consider several types of businesses who have been familiar faces in the IT industry: services businesses, system integration and software component businesses. Frankly we don't have an optimistic message for any of these. The service people are facing the fact that the Internet is making it possible for small information suppliers to offer products with very little barrier to entry. The systems integrators are facing the fact that the price of software is being driven down. Thus the cost of a customized job done is looking relatively more expensive. The software component businesses have some problems, much of this is being unbundled and the price is dropping.

3.4.2. Will businesses give up on integration?

Robert Worden: This is an important question. It is a common idea that anyone can do software, but large integration problems are bloody hard aren't they?

Mike Lesk: Everybody THINKS they can do software.

Mike Braude: But I would suggest that integration is the out word and people have given up on that concept. At least in these commercial enterprises we are not going to see a unifying architecture. There is not going to be any overall design because people have been trying to do that for two decades and they haven't got anywhere.

Les Belady: I don't believe that.

Mike Braude: Well most people out there believe that integration is a marketing term that vendors offer to get you to but their products. But look at the reality out there. You've got relational databases, IMS databases, you got stuff under ACCESS and what is the meaningful plan to integrate that? There isn't any, because as soon as you come up with one there will be some new database or standard.

Peter Wharton: I think this means the continued development of such concepts as 'framework' architectures, the use of models within which standards can be defined, coexist and evolve, and the formalisation of many concepts such as encapsulation, modularity, inheritance, use of early and late binding etc., in such ideas as client-server architectures and object oriented technologies.

Overall I would like to see these being regarded as a **"don't care capability"**; that is methods and tools that don't rely on the exact implementation of the same interface or protocol, and are flexible enough to deal with change in a cost-effective manner. These may appear in many guises; change management architectures, integration mechanisms, reverse engineering tools, and so on.

Robert Worden: Still, giving up on integration is an interesting point. For example is AI going to be the glue that provides a veneer of integration?

Mike Lesk: Well the idea of integrating all the data in an organisation is going out. And the attempt to unifying all the programming in an organisation is going out because it is being done all over the place.

Les Belady: I'm sorry, giving up on integration means what?

Gill Ringland: I think it means people are giving up on the on the idea of a single corporate data model - the central planning of corporate data. And instead they are saying, 'let's just make a jigsaw'. As in the past, the linkages between islands of organisation will continue to be made by people. And they will continue to develop software tools to help automate those linkages.

Mike Lesk: So the buzz words in the future will be modularity and interconnection.

Robert Worden: Here I see an opportunity for AI to have a role in helping talk between the different data models. Because we know a lot about those systems because we built them.

Mike Braude: And what do you see there?

Robert Worden: Well if we talk about telescript on the net and agents going out and doing things, okay. And if all these agents work only with unformatted text, okay. But if those agents are going to access structured data it seems they will need some sort of rules or heuristics to be effective and AI could play a role there. The domain of IT is a somewhat structured world and I think that AI has some techniques that could be useful here. Businesses will find places where they can add value by getting two of these systems to talk to each other.

Remi Bourgonjon: Some people define AI as a solution desperately looking for a problem.

Les Belady: Probably the right model is that AI is the stuff out there we don't know what to use for. As parts of it mature and get incorporated into useful things, those parts are not called AI any longer.

3.4.3. What is the difference from 20 years ago?

Les Belady: If I put up a list of what we want from systems, the words haven't changed. We want:

- integration
- flexibility
- easy (re)configuration
- multiple levels of modularity
- some frozen components/interfaces.

But the meanings we attach to the words have changed because we have a different external and technology environment from 20 years ago.

We didn't foresee then the widespread use of networks. We didn't foresee the widespread availability of components. And we didn't see how to put intelligence on the network - the concept of agent has been very powerful.

3.5. Software and monopoly

3.5.1. Software as a 'natural monopoly'

Mike Braude: One think I would like to know is whether you think the break up of the Bell system was a good idea?

Mike Lesk: I think it was a stupid idea but of course I have a bias here.

Mike Braude: I thought it was a stupid idea too, but I wanted to know your opinion as an insider. In the same way I think it would be bad for the consumer to have IBM or Microsoft broken up.

Mike Lesk: There is a guy named Michael Noll at the Annenberg School who is writing papers trying to prove that, on average, the break up of AT&T raised prices. The fraction of revenues that subsidize local calls has actually increased in the ten years since the break-up, counter to the original goals. The shareholders of AT&T are clearly better off and the US trade balance worse off by about $10 billion per year without any compensation.

Mike Braude: It might be interesting for this group to look at the joint action by Novell and the US Government against Microsoft in this light. One wonders if this could drive the price of software up.

Mike Lesk: The difficulty is that software is a natural monopoly business. In economic theory, natural monopolies occur where there are large economies of scale. Clearly this fits here because reproduction costs of software are zero. In addition, customers and other manufacturers of related products want a common base. The people who make printers want standard ports and paper sizes just like they want a standard voltage for the power supply.

If no one is allowed to have that monopoly, the result could be what we have in the airline industry, a series of price wars and bankruptcies and what not. I think that airline deregulation, on balance was a mistake. I didn't used to have to be an expert in airline reservations, now you can't manage. You used to be able to take a ticket and use it on any airline, now that is gone. The airline industry has lost more money in the past 4 years than it made in the combined previous 50 years. So I'm afraid that breaking up the Microsoft monopoly will just lead to a lot more confusion in the consumers mind.

Robert Worden: The cost of bringing software to market is rising rapidly. If this continues, it is a factor that will push toward a monopoly situation.

Mike Braude: So let's put out this hypothesis: each monopoly contains the seeds of its own demise. Partly this is because each monopoly spends too much effort defending its own technology. (Reference: Innovation by Richard Foster)

3.5.2. What are the next monopolies?

Gill Ringland: So now we have another question. IBM had the mainframe market monopoly 30 years ago. Microsoft has the PC monopoly now. What will cause the seeds of its destruction and where will the next monopoly be built? Will something, from perhaps the games industry, replace the Microsoft desktop monopoly?

Remi Bourgonjon: I don't imagine a monopoly as broad as the that of IBM. There will be many new domain-specific formalisms and perhaps 'monopolies' restricted to these.

Mike Lesk: This reminds me of a rule for buying software: all you need to know is the release number. Never buy the first release, never buy something like 'DBASE one'. And never buy a release number over ten. (Laughter)

Robert Worden: Well regarding new languages, I'm afraid we are out of luck. The world is moving toward C++ and it will last the rest of this decade, perhaps as long as COBOL has.

Mike Lesk: I think, with some luck, C++ will collapse of its own weight.

Robert Worden: I would dearly love to steer us away from C++ but I don't see any other industry standard.

Mike Lesk: (Laughing) I really like this because it contradicts the previous point. The previous point says, the business is a natural monopoly. Now we conclude there will be no single dominant language.

3.5.3. Will it (or they) come from entertainment?

Robert Worden: I want to go back to answering Gill's question. What will be the next framework? Could the next thing be telescript, some type of API?

Mike Braude: The elements of the next framework are networks (wireless), mobile computing, and entertainment based software. Also it is likely that the funding will come from the entertainment industry.

Mike Lesk: I'm not certain that the new framework will come from the entertainment industry. Does Disney have a research lab?

Gill Ringland: No but Sony does.

Les Belady: Excuse me, but I think we are over emphasizing this entertainment industry.

Gill Ringland: Now we call it entertainment, because we think it will be based on an interactive TV interface. Of course we will do many other things like buying insurance, making travel arrangements. Perhaps we should say consumer-oriented instead of 'entertainment'.

Les Belady: Do we have any idea of where the next 'killer application' will be based? In the past there was often a sequence of first military then business adoption and finally the consumer market. Is this always the pattern? Who do you sell to first? Business or consumers?

Mike Braude: Teenagers now have money and their spending patterns could drive the next killer application. This is a market that can generate fads. I can imagine Charles Barkley or Madonna in a commercial talking about how they use some piece of software.

3.5.4. Past and potential future technology markets

Gill Ringland: The track has come from several directions, I can think of examples - for instance the initial large penetration of telephone, FAX and photo-copying was in business; radio, TV, Polaroid camera, PC (hobbyists - remember the early Apple market) and CDs in the consumer market, and the military developed the jet engine and radar.

Mike Lesk: In broadcasting there was an early belief that TV would be educational and uplifting, but that went away early. Today we have that hope for the net but I recently saw a report that 8% of the Netnews traffic was related to pornography.

Les Belady: So it seems from an initial, superficial review, there is no obvious rule that emerges. The initial market for new technology could be an any segment. So, what new 'killer applications' do we think are likely in software?

Mike Lesk: This is difficult. If you asked us 10 years ago I would not have thought screen savers would be a big hit, but everyone now has one.

Mike Braude: One area will be related to sales-force automation. In many companies, half of the employees are in sales and various sales support functions such as order entry. Selling platforms to those people will be a big business opportunity. The emerging technologies, wireless communication, portable devices that could provide video displays and database access offer competitive advantage.

Mike Lesk: Another idea is that we put disks in a photocopy machine, or a phone, and then you can recall things if you need them later. This might be a service that a lot of people want.

Les Belady: Another factor is that there are shifts in the constraints that define products and businesses. We should consider what changes of this type are occurring or are likely to occur. For example, several years ago it seemed there was a shortage of available radio frequency bands but more recently there have been technology changes that allowed us to go much further. This is just an example of what I mean be shifts in the constraints. What limits do we perceive today that will change?

Mike Lesk: One observation is by Negroponte that in a few years there will be a reversal of broadcast and wire-carried information.

Mike Braude: There are large constraints in the areas of liability and intellectual property rights but I think these issues are being addressed by the Environmental sub-group at this conference.

Mike Lesk: It is an open question whether the courts will place different requirements on software than on printed materials. For example if I write a book that recommends that you buy stocks, you follow that advice and lose your money, there isn't a liability that covers you as a book buyer. The liability on the book is that all the pages are there and the cover doesn't fall off. However let's suppose someone sells you a computer program that recommends stock purchases and you lose your money and later it is discovered that the program was written by some 13 year old who never had a course in economics. Most of our laws were written for a different era.

Robert Worden: One bottle-neck that I think will remain with us will be that the size and complexity of some software projects will always grow to the limits of our abilities to manage them.

Mike Lesk: Oh, I think some of them will go well beyond that limit. (Laughter) But this takes us back to the problem of defining software limits, which is why specifications are so difficult to do correctly. Consider that if you talk to a 747 designer at Boeing and say 'I want you to add a swimming pool to the aeroplane' he can make calculations and tell you, no, it is too heavy, the plane can't fly. But if go to a software engineer and ask for a project of similar difficulty he will say, well, if we add another megabyte of RAM and a few more programmers... It is just more difficult for us to see the limits in software.

3.6. Impact of the Net

3.6.1. Changes in ways of working

Mike Lesk: There are 26 megabytes of new Netnews each day. The number of subscribers on the net doubles each year. There is a huge potential for much greater volumes of information but we don't have a good economic and legal model to allow publishers to put out stuff without loss due to theft and copyright infringement. There are now more humanities journals available on the net then there are technical journals.

Les Belady: One of my concerns is that many professionals will spend more time learning about their areas by reading or conversing on the net, rather than having hands on experience. This could lead to a shift in the type of understanding they will acquire. Consider the analogy that in America boys used to spend hours fixing cars. They learned how to diagnose problems and change parts. But we may fail to develop some understanding and control of the things around us? Will we ever be able to reprogram our own burglar alarms or carburettors?

Robert Worden: We will just plug it into the net to get an expert diagnosis.

Mike Lesk: If you have a car phone, your car will be continuously linked to the net.

(Laughter)

Remi Bourgonjon: And if you failed to learn to make certain patterns on your sewing machine there should be no problem. As we heard this morning, today's sewing machine now has an RS-232 port for down loading the necessary instructions.

Mike Braude: The PC changed the way that individuals worked. The next logical question is whether access to the Net and groupware will have a similar impact and performance improvement for people working in groups.

Les Belady: When we talk about the Internet we talk about two separate aspects, one is that I can access all kinds of databases and the other is that I can talk with all of these people and experts. And these are very different types of use and it parallels how we seek information in our everyday life.

Robert Worden: That is just the sort of thing that will change in a few years time. Suppose you are browsing some document the net and their is a button advertising that someone on the Net is offering a live consulting service on that topic. You just press the button and up pops a small image and you have a short conversation.

Mike Lesk: There are people who think this will be a great money maker but I am sceptical. I am also very sceptical about video conferencing. There have been lots of studies over the years and they don't fill me with enthusiasm.

Robert Worden: Well here is another set of contrasting opinions to write up.

Mike Braude: Have PCs really changed the way we work?

Mike Lesk: Yes, we spend about ten times as much time writing documents.

Mike Braude: But has it really changed the way you go about your job?

Mike Lesk: One change that network interaction could produce has been suggested by Garrison Keillor in his piece called "Rise of the Shy". He says it will lead to a reversal of the historical situation where the loudest, prettiest or biggest have dominated discussions. Instead those who have the best writing skills - for example, those who were shy - will now have the biggest influence.

3.6.2. Effect on digital property

Mike Lesk: The network has forced into prominence a number of legal issues to do with intellectual property rights: the basic problem is that networks are global but the US, Japan and Europe have different methods for protecting software. It is not clear how this is going to be resolved, the suppliers are often US based but this isn't necessarily the deciding factor.

Charles Simonyi: We have seen a flurry of patents and these are often not helpful to the business as a whole.

Mike Lesk: The network raises the question of liability also - how can software and network suppliers be held to warranties?

Remi Bourgonjon: The look and feel suits conflict with the trend which we think is helpful towards de facto standards.

Gill Ringland: I'm not sure if there are answers to these but Jessica [Litman] is definitely the person who can illuminate the problem and look for parallels to find a solution.

3.7. Research topics and process

3.7.1. Areas for research

Mike Braude: The main problem that we are hitting is the problem of how organizations - we deal with the Fortune 1,000 companies - how can they possibly accept change, given the momentum and culture behind the ways they do business; this rather than the technology is the key problem as we see it. It's a problem for psychologists rather than technical people.

Les Belady: I wouldn't expect those guys to see the complexity, I might try an anthropologist (laughter) or sociologist.

Robert Worden: Another topic - I would argue that the needs of our "intermittent" programmers - the 200 million - are going to be different in many ways from those of professional programmers. They may well work in a number of different environments depending on their current task, and they may well need to do a mixture of browsing, retrieval, computation, display and generation of documents that involves an enormous amount of glue and interfacing. If you gave it to us as a project we would solve it one way, maybe intermittent programmers need a whole different approach.

Mike Braude: And I think that in the supply of software to those intermittent programmers, a whole new set of distribution channels and techniques open up - not only the economics and but also the motivations and criteria of the buyers within the business model changes.

In fact the whole area of electronic commerce - how does it work in practice, what are the legal implications of digital property, seems to me to need analysis before the levels of trading - and also the law suits - get to crisis levels.

Robert Worden: As part of electronic commerce and life on the net, my belief is that one of the applications of AI must be smart agents. Doesn't the research of Mario [Tokoro] start to tackle this - this might be one of the areas where computer science has to lead.

Remi Bourgonjon: I think this whole area of frameworks for different domains is important. We are needing to be able to take components from different parts of the IT industry and developed under different frameworks - it is not quite the same as the traditional standards activity - it is more like finding useful (logical) levels at which to provide interfaces.

I see an increasing number of domain specific frameworks or platforms, for instance:

- entertainment creation /authoring
- visualisation
- information retrieval

where the API's and languages and tools are different but the completed components may need to work across domains. Frameworks evolve - how can the evolution be within "safe" limits?

[See also discussion of standards in Chapter 2]

Gill Ringland: Considering HCI, we talked earlier about the intermittent programmers. For these in particular, could we be looking outside the traditional IT industry to understand what is and what is not intuitive. We have just the one paradigm, the Windows/icons paradigm. Where will alternates come from? Experiments would seem to be in order here.

And the bête noir - how do we measure whether we are getting better at software?

Les Belady: Let me say not lines of code per man year before any else does.

Mike Lesk: You need a theologian to attack the problem.

Gill Ringland: Let's keep it on the agenda. Without some sort of answers from the professionals on this, we're somewhat handicapped in deciding where to put our efforts.

Robert Worden: Can we hope that the Technology group have covered it?

Gill Ringland: Many of these research topics seem to require interdisciplinary approaches. Can we identify what we think are potential lead organisations? For instance, in the case of user model studies, the lead organisation could be an instrumentation company with design, computer science and psychology academic departments, or alternatively consultancies providing supporting expertise.

Table 3.1 summarises the results of the discussion.

Research topic	Potential lead specialisation
Acceptance of change (organisational dynamics)	Psychology
Glue technology and methodology for heterogeneous systems (for use by "intermittent" programmers)	Computer Science
Software distribution channels	Marketing and Sociology
"Electronic commerce" for software distribution and sales	Law, Business School, Computer Science
Smart agents	Computer Science
Intellectual property and value information	Law and public policy
Frameworks for animation	Industry
Person-machine interface, HCI, user models	Industry, Design and Psychology
Determining useful metrics for software development	Industry and Theology

Table 3.1. Areas for research.

3.7.2. Roles in software research

Gill Ringland: The model coming from that discussion is that the process for IT research may be different from the classic model, such as might be found in chemistry, in which new ideas originate in the academic sphere then migrate to industry where they may be implemented, i.e. scaled up into industrial size process and introduced to the market That model does not seem to reflect much of what happens in software.Examples include relational databases originating from Ted Codd at IBM and the seminal role of Xerox Parc. And it does not seem sufficient today - with the tremendous rate of change in uses of IT; we could expect some innovation to be driven by new uses of IT.

We seem to be at a stage for experimentation and evaluation, which tends to be driven by industry (or government) for two main reasons: firstly, it is risky - a realistic estimate based on venture capital statistics would be that 9 out of 10 project "fail" in that they do not lead to innovation (see Figure 3.3, below) i.e. introduction to the market. In the current state of evolution of software and IT our belief is that this proportion is not unreasonable. Though the concept of "failure" is relative - the work at Xerox Parc was a commercial failure for Xerox, and the first Apple systems incorporating the technology also failed. But the work of course has been immensely influential in defining the PC interfaces of today.

Secondly, experimentation and evaluation tends to require larger, often multi-disciplinary, projects.

Yet it is difficult in many large organisations to manage this level of experimentation, both culturally and financially. Culturally because bureaucracies act to try to avoid failure.

Bill O'Riordan: Financially because the hardware vendors have traditionally funded this research out of their hardware margins - which as we know have shrunk.

Mike Lesk: As I understand it the venture capital industry is not developed in Europe to the same extent as in the US - doesn't this mean that there's a question of where funding for software development is going to come from in the EC countries?

Gill Ringland: There is no easy answer to this but it may come from new applications in other industries. As the forcing applications move further from the traditional IT industry, this means that the role of academic research in assessing, generalising and transplanting concepts, becomes much more crucial. The lessons of the experimental projects will probably not be captured by the active participants in a way which allows them to be re-used. So academic research is crucial to get the concepts in a form to facilitate technology transfer.

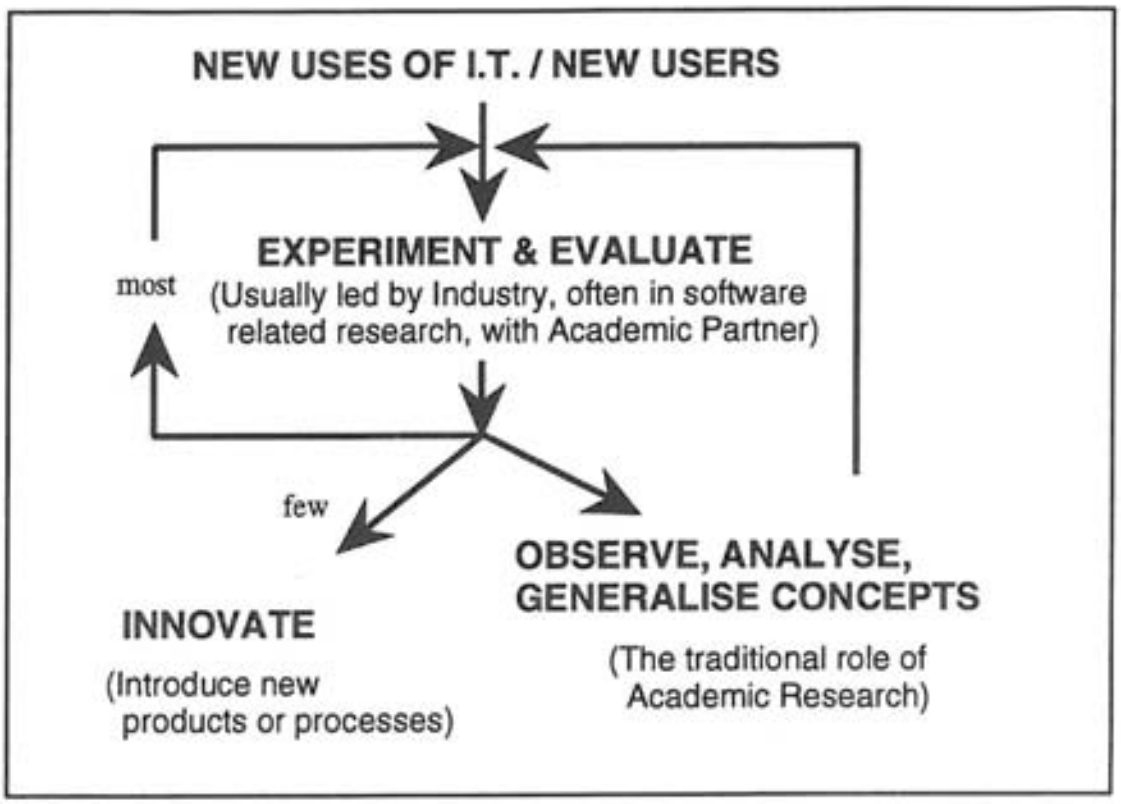

Figure 3.3. Roles in software research.

Chapter 4
Technology

4.1. The contents of this chapter

The technological advances foreseen in the near future will have a major impact on the discipline of software engineering. The working group began by discussing a general model in which levels of programming, and the respective types of programmers, could be represented. Subsequently, some time was spent analyzing the impact of new technologies, namely networking and hardware, on the discipline of software engineering. Next, the structuring of software, in terms of its components and the means by which these are put together, was discussed, giving particular emphasis to the form of future components. Then the group spent some time discussing various issues directly related to software engineering. The final topics discussed were issues related to the human-computer interface, possible new computer applications, and research requirements.

4.2. Programmers and programming

Brian Randell: There is a spectrum concerning type of programmer - from people who are very much professional programmers and know it, and who are paid by people who know it, through to people who do not regard themselves, and are usually not regarded by anyone else, as programmers.

Charles Simonyi: There is a separate spectrum concerning type of programming - from VCR programming, say, at the one end, to LISP hacking at the other. Scientific programming is kind of in the middle.

Gilles Kahn: The number of people who at one point or another have to get involved in some kind of programming activity is very large but the number who design programs that will eventually be used by somebody else is rather small.

Charles Simonyi: Between one fifth and one tenth, say.

Gilles Kahn: Among the people who are doing some true programming and not just parameterizing, those who are going to compose programs that are used by others form a small fraction. Where do the largest amounts of code reside today? It used to be that operating systems and similar things were the largest programs - now it is end users who have the largest amounts of code. I know for example an oil company that has seven million lines of FORTRAN.

4.2.1. Evolutionary cycles

Charles Simonyi: The way you entice non-programmers to program is to take a popular application and make it programmable, for example Microsoft WORD. A spreadsheet is another good example, and not just because of the equations that are in it.

Gilles Kahn: By such means you can parameterise and adapt an application to your own environment.

Charles Simonyi: Yes, parameterization is the first step on the slippery slope.

Brian Randell: An interesting point is the notion of cycle and progression. We start from a system which is very rigid in what it can do. Very often additional flexibility is then provided by allowing parameterizing or customizing, leading to a more complex system that eventually is recognised as needing new abstractions. A new abstraction is found, perhaps incorporating domain-specific knowledge. Sooner or later this is extended to allow parameterizations and customizations, and then the evolutionary cycle is closed (as shown below). In some cases with these abstractions we spread into different communities, and move away from any general realization that what people are doing is actually programming, with all the good or bad consequences that this involves, e.g. related to the perhaps unmastered complexity that they are trying to cope with.

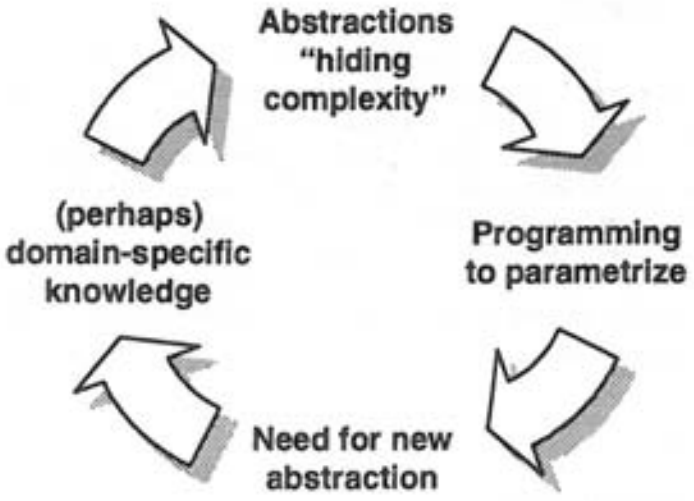

Figure 4.1. Evolutionary cycle.

Gilles Kahn: It is not clear to me that the methods that are currently used for producing safety-critical software, for example for nuclear reactors, or for cars, will evolve and scale up adequately to match future expectations. On the contrary, for real-time systems I think we are at a "fracture point", a bit like it was in the 70s, when operating systems began to be built in higher level languages.

Brian Randell: The evolutionary cycle that we have talked about is one in which your "fracture points" are introduced by introducing abstractions. What you are indicating is the danger of having a growth in size and complexity without adequate abstractions.

Hervé Gallaire: Abstraction is a very powerful concept; it can be asserted that it has been the major contributing factor in the evolution of software engineering; object-oriented technology, the latest addition to the software engineering framework, has been promoted largely because it supports abstractions to a greater extent than other programming technologies. By their very nature, abstractions allow us to adopt a view of a system that brings salient issues into focus while hiding issues of less importance. We have used this to achieve "information hiding" or "abstraction boundaries". The abstraction and its modular implementation behave as black boxes, irrevocably hiding certain choices behind the abstraction boundary; this is indeed what users want, provided that the abstraction reveals, above the boundary and through its behaviour, what they are interested in, exactly. If it does not, and it rarely does, users of the abstraction, that is, programmers or designers, have either the choice of programming around it, leading to obscure and redundant code, or have to redesign and redevelop new abstractions, defeating the major purpose of using abstractions, that is, to enable reuse of design and of code.

Brian Randell: If you take almost anything that provides a great deal of flexibility you will find that the spectrum of levels of effective use is very wide - there will be quite a number of people who will learn and use just a first few features. My own use of Microsoft Word is a case in point.

Charles Simonyi: We are working on this issue quite hard, and we have these things called Wizards that will walk you through procedures that you do very seldom, for example printing envelopes. There are a fair number of free parameters there. If you want an envelope to be printed you get something like a training course, but which is not a training course in that it lets you do the job, but does not preach to you. So you will have no problem with it, I guarantee you!

Bill O'Riordan: I have to hand my watch to my daughter when it needs adjusting.

Charles Simonyi: That is because your watch is too dumb. It requires incredible amounts of power to make things simple.

Gilles Kahn: The software system must provide some sort of a modelling system for the user to parameterise, and this modelling system is not strictly necessary for the system's functioning. It is there just to interface and parameterise for some specific purpose; this has to be handled by some sophisticated man-machine interface technique, in order to facilitate the interaction. Another point, is that there is a general move to replace as many programming activities as possible by parameterization activities. This has been a general trend. For example, parsing generators are just instances of replacing programs by data. And the last point is that, has to do with composability. When we compose an application from a number of components, they will have in general their own parameterization facilities but we will have to understand how to perform the global parameterization for the assembly, and clearly we have to hide the fact that the application was built from parts.

Charles Simonyi: You are correct in describing the historical progression of parameterization. However, at the same time parameterization is regressive, that means that if you parameterise things with respect to n parameters you are really removing a lot of degrees of freedom that people will miss, and that is why is important to have some kind of symmetry, in the sense that whenever you have parameterization you have to have generalization, and vice-versa. Your last point was evidence of the backlash of parameterization: when you compose these things, you have these free parameters that have nowhere to go. The problem is how do you propagate these to other levels of abstraction, unless you have a general capability. At Microsoft, for example we are using Visual Basic as a general escape mechanism at each level.

Tony Temple: I think programming evolves to a point when it ceases to be programming. In my opinion if we define things in terms of the system, for example in accordance with its philosophy of sequencing, then we are probably programming. However, when we start to define things in the users terms then we are not programming. Now people are defining things in their terms which have nothing to do with programming whatsoever. I think the difference is whether we do things in system terms or user terms. Recalling the envelope application, I think in this you are working in user terms, so you are not programming, because it is based on the user's model of that application.

Hervé Gallaire: As soon you start talking about composability, you want to do composability in user terms and that still is programming because we do not have very good ways of doing composability today.

Tony Temple: The more you are in the user's domain, the less you are programming.

Hervé Gallaire: Composability has to do with programming, by definition.

Gilles Kahn: The tendency in the future is for composability to become more important due to the availability of distributed systems.

Charles Simonyi: I also think the need for GUI will increase. You can have a GUI interface in things like watches, telephones and interactive television. The basic characteristic of GUIs will be to minimise the user's need to remember possibilities, and maximise the ability to help him/her by presenting alternatives and being robust. Many of the programming sequences that we come across are extremely fragile with respect to errors that you commit in the middle of sequencing.

Mario Tokoro: Programming could become closely related to artistic expression. Very close to the concept of parameterisation, but is different in that a program for an information provider might consist of editing a couple of things like video, voice and so on, and presenting what one feels is right. This can be represented by visual programming, which allows one to define a program graphically. In other words, programming will become more artistic rather than more mathematical.

4.2.2. Fragmentation and spreading

Brian Randell: The progression is to invent new abstractions from the parameterization of other abstractions. There is a sort of moving target, which leads to there being different levels of programming, ranging from ones which are not evidently programming to ones that are very evidently programming but in some domain where people might have not recognised that what they are doing is really programming. Through this progression, programming becomes more diffuse, in the sense that it becomes fragmented into an ever greater number of distinct application domains.

Bill O'Riordan: So what then is meant by the term "professional programmer"? People who encode a specification (very precise statement of requirements), or evolvers/designers/creators of very efficient algorithms?

Charles Simonyi: A professional programmer knows, and can explain, what 'assignments', 'ifs' and 'loops' are. By saying this, I'm trying to bound the upper end of a non-professional programmer.

Mario Tokoro: A professional programmer produces programs to sell.

Tony Temple: In an IT department of a company they do not sell programs, they use them.

Gilles Kahn: As each discipline developed special techniques and tools, what people in each different domain term programming, whether it is the COBOL world or car automata, has become more and more different. Intellectually, it is the same kind of ability and the same style but technically it is different. This causes a fragmentation and spreading of the activities performed by programmers. Some programs are done by people who do not consider themselves as programmers.

Hervé Gallaire: Because of the spreading, people have to be both software and domain application competent.

Charles Simonyi: And non-programmers are doing programming-like things. That is the fringe of the spread!

Bill O'Riordan: Somehow or other we in our business need to equip our staff with a knowledge and skills portfolio which will allow then to manage and control most of the major technological changes anticipated over the next decade, if we are to compete responsively and profitably. We need something more than "movers and shakers", we need "movers, shakers and paradigm shifters" to coin a phrase.

Increases in the rate of change of information technology have decreased the effectiveness of experience as a teacher. Experience is too slow. Trial and error require more time than is currently available between changes that require a response. The lag between stimulus and response brought about by reliance on experience permits crises to develop to a point where we are forced to respond to them with very little relevant knowledge. An increasing portion of responses are made out of desperation, not out of deliberation.

Hervé Gallaire: Do we really need software engineers who are purely software engineers, or do we need people who are software engineers but, as well, understand specific domains, whether it is control of devices or an application domain like combinatorial problems? Don't we need people who understand both. And because we have better training, and people start younger, it is possible to be a good a software engineer and get initial skills to be more effective in the application world, as opposed to being just a software engineer.

Brian Randell: There is a much bigger need for people that have twin skills, though we can think of those people who are present-day software engineers as also having a twin skill in that their application area is the computer itself.

Gilles Kahn: About those programmers in the diffuse or embedded computing, to me it is not clear at all that they will accept the existing models of computation. I would like to hear something about finite state automata and control mechanisms.

Hervé Gallaire: Such programmers will program at both a high and a low level. They will manipulate automata with the notions of states and transitions directly, as opposed to writing C code. This happens already because we have hardware which directly embeds automata.

Mario Tokoro: According to my observations, the necessity of programming for the real world, such as real-time programming, networking programming, communications programming, and so on, will become much more important in the future. Unfortunately, at this moment, this type of programming is not well taught in the universities. My question is whether this kind of real world-related programming goes as far as the end user?

Tony Temple: I do not believe that we have just a single profession in software engineering. So I think a whole set of disciplines are required, people have to be specialise in these more and more. For example, we have various sorts of programmers, and not many programmers make great designers, or user interface designers. There is no such thing as a general software engineer, any longer.

Charles Simonyi: Some of these issues are separable. Algorithms and user interfaces go together. But graphic design is rather easily separable and there what we have to communicate to the programmers is humility.

Brian Randell: There are two points. One is the notion is that we need people typically with a couple of skills, and then we need a lot of different such people. Of course, in any general engineering we find quite a lot of that, and to what extent there is a useful generality that is still worthy of identifying is an issue that is very difficult to predict 10 years ahead.

4.2.3. Legacy programmers

Brian Randell: At each stage of the cycle you find there is a group of people in a generation that have learned something, but who then regard that as a valuable skill that they can go on using regardless of changes.

Charles Simonyi: At the level of society as a whole, this does not matter.

Brian Randell: You can go on for a long time with a lot of people hanging on to a particular skill. To what extent do the evolutionary cycles evolve via younger people and leave older people behind?

Tony Temple: A lot of programmers cannot transit from one generation of programming to the next. The different generations in programming do not provide the kind of continuity necessary for some people to jump into the next generation.

Brian Randell: This is just an example of the standard human problems of adapting to change.

Charles Simonyi: Legacy programmers are affected by down-sizing, when the system has to be rewritten and the environment is changed. At least in the States such down-sizing is the biggest thing that is going to happen, as business moves away from the mainframe, and everything gets rewritten in C in distributed fashion.

4.2.4. The role of theory

Hervé Gallaire: The root of the comparatively low use of mathematically-based modelling techniques in today's practice of software engineering, as compared to the use of abstractions, is clearly due to the fact that we haven't had coherent, integrated methodologies for developing complex computational systems based on modelling ideas. Generally applicable reasoning techniques have been emerging only slowly, and mostly in well understood areas of mathematics, such as type theory or linear programming. This is not a criticism of the whole field of theoretical computer science, which aims in part to provide such tools, but it is the reality of software engineering practice, even if in areas such as testing the use of reasoning capabilities is slowly becoming a reality for safety-critical application domains.

Bill O'Riordan: Theorists have a great role to play: they simplify algorithms so that people can learn quickly enough, and apply the knowledge before becomes outdated. Theory will become terribly important in terms of programming in the future, so we actually can move fast enough before experience becomes irrelevant.

Brian Randell: As regards theories of finite state automata, I remember Doug Ross of SoftTech many many years ago saying that eighty or ninety percent of computer science would eventually be done, and very usefully done, using finite state automata, once we learned how.

Hervé Gallaire: I know of someone at Boeing who is working on airplane stabilization systems using pure automata theory. It is incredible what is done now with finite state automata theory.

Charles Simonyi: The key thing here is that such things are domain-specific.

Gilles Kahn: Theoreticians see these as automata, but the people in industry always think in terms of block diagrams. So even though the programming style is domain-specific, there still is a gap between theoretical training and people in industry who use much more graphical ways of expressing themselves.

Brian Randell: A great amount of finite state automata theory was very successfully buried into UNIX's tools and filters, and so allowed a lot of people to work on a higher level and produce very effective programs. They were living on the fruits of that theory, but did not need to understand it themselves, and that is the most successful way that such theories are transferred.

Hervé Gallaire: By using theory we are able to abstract much better from a problem so people are able to understand more quickly the generalization of the problem, instead of learning first its specifics. As an analogy, we first learn about sorting before learning all the possible algorithms.

Brian Randell: Theory is very important, but more for the benefits that can be produced or used by other people, than for having a set of theoretically oriented programmers who use theory heavily. In summary, the benefits of theory are more indirect than direct. I am against the thought of considering programming as mathematics, and the training that you need as simply a mathematical training.

Mario Tokoro: A limitation of the current theories is that they are very static in terms of the system analysis.

Brian Randell: A lot of the early theories of concurrency were about concurrency amongst a fixed number of players, for example, but the theoreticians have moved on to deal with more dynamic situations.

Gilles Kahn: I agree - people want to be able to capture in mathematics very complicated dynamic behaviour in some form of equation or calculus these days. I don't think that theoreticians are insensitive to this, though sometimes they are a little disconnected from what the real practical problems are.

4.3. Influence of hardware developments

Brian Randell: Of the predicted developments in hardware the ones that will have most impact, and impact on the software world, are those in communications - these will bring whole new application areas into existence, will place new demands on software design, with new problems arising, for example to do with naming and addressing.

Charles Simonyi: These communication facilities provide a complete new sort of scalability. Before, if a company needed more power, they upgraded from a Model 50 to a Model 65, say - today they buy a couple more PCs.

Gilles Kahn: There are some other factors coming from hardware, for example we will want to use images and voice, that will have an impact on the network connection. There are two needs, one is the increased CPU power which we believe will come, but the network infrastructure is a different story, an Ethernet of 10 Mbit will not stand for the future needs, and FDDI is not going to make it.

Brian Randell: My impression is that network bandwidth possibilities are increasing even faster, if we look at growth curves and the predictions for the next 10 years. With the next bandwagon, which is ATM, the ability to send pulses down fibber is quite incredible.

Hervé Gallaire: You may be get to 1 Gbit, but that is all. It is only a factor of 100 with respect to Ethernet, and a factor of 10 with respect to fast Ethernet. The main thing is that the cost to use this bandwidth will be very low.

Gilles Kahn: My view is that communication speed increases occur in steps, because you are always limited by the narrowest pipe. Until this pipe is improved, you have no effective increase in communication speed.

Brian Randell: What matters is the "narrowest pipe" within the relevant set of machines of interest.

Charles Simonyi: The cost of network bandwidth will fall dramatically, perhaps becoming essentially free. By this I mean that the amount of money spent on hardware will go up, but the cost per Mbit per second will fall. You cannot expect to pay the same for your phone link when you have a 10 Mbit per second link into your home. We may even start paying for a network service, rather than by consumption. An analogy here is water in the home, where you do not care how much you use (though this varies from country to country).

Gilles Kahn: This comment that the cost of communications will reach zero is counter-intuitive to me, when I look at the budget of my Laboratory. Over the last ten years I have found myself paying far less for computers and far more for communications. I use far more communications. The cost per bit may have decreased considerably, but the total bill has increased.

Charles Simonyi: It's like water. Of course if you look at your building budget there are a whole lot of copper pipes and faucets, but you say that water is free - that means that you are not worrying about its availability, or your future needs, you don't make trade-offs of X versus water. What we can say is that what we are communicating today will cost zero tomorrow. We will have to pay for our greatly expanded future needs.

Hervé Gallaire: The business model may change. We will not pay for the communication, we will pay for the service.

Tony Temple: The network is giving a data access performance which is equivalent of going to the main department file, which is interesting because I can get data wherever it is with the same speed. I think we will soon have this not only in our physical domain but across domains, even world-wide.

Brian Randell: There are some experiments based on the fact that networking is getting so fast that you need a different interface to networking. One approach which is challengeable, but very interesting, is to have a shared memory abstraction which completely hides the networking.

Brian Randell: A critical issue with the steady growth we are assuming in a number of technologies, is whether these growth rates are sufficiently different for existing balances to change significantly. The sort of issue whereby networking becomes a possible substitute for disk, even for memory - those change the designer's world rather a lot. A lot of changes as to what are the important problems and how we should solve these problems are to do with such balances.

Charles Simonyi: The reason why you want to replace the disk is not to save a few dollars, it is good to replace the disk for these reasons of backup and better, but not for cost reasons. Rather it is for reasons of reliability of information, so that you are referring to the latest data, not to some copy of the data that was carelessly copied there.

Mario Tokoro: Information itself will be very different to what we think about now. It will not be like a copy of mail, or of programs or data, but instead will be something like real-time on-demand video, music, news and so on.

Tony Temple: Another thing that we have to consider here is that the reduction in size, dimensions, weight, or whatever, that is happening at all levels will change what we do and will have an affect on the applications we are going to tackle. The point is that whenever you change the scale of things you are introducing new applications and opportunities that were never thought of.

Gilles Kahn: Whenever we talk about things getting smaller, we should also talk about the dual thing which is the system becoming more global. For example, to have a small cellular phone which allows you to talk with Australia, these two things go absolutely together.

4.3.1. Consequences of networking developments

Brian Randell: Okay, I think we are fairly clear that communications developments will allow us to build vastly more complex systems, essentially by interconnecting systems. However, the complexity and fragility of these systems will be a major challenge. Such distributed systems provide both dependability problems and dependability solutions and if you are lucky or well-organised you get the latter rather than the former. A lot of the difficulties experienced with distributed systems are because they can consist of a great set of interconnected single points of failure, many of which you have not identified beforehand.

Gilles Kahn: With this incredible high bandwidth on the network, we have to have more computing power in workstations to swallow all that is coming, so even more of the computing power will be used in underlying work, such as compressing and uncompressing. This will be a bottleneck.

Charles Simonyi: Not at all. Parallelism will solve the problem of bottlenecks in a workstation.

Gilles Kahn: Do we have an opinion as to whether distributed databases will be the final doom of mainframe computers?

Bill O'Riordan: Enterprises and large corporations will not distribute their databases, physically or logically. Once you have distributed the data and something goes wrong, you can lose the business. It is well known nowadays that when you buy a company the first thing you get hold of is the database; the data is the most important thing.

Tony Temple: Data will be distributed all over the place. I mean by distributed data the sort of situation where you have got personal data on your own system, you have got departmental data in some server in the room, and you gradually get up to a level where you will have global sharing of enterprise information, you will have data at all levels.

Hervé Gallaire: You will also have virtual databases, like virtual organizations, so that what you need might not be in one database but spread across many different databases.

Charles Simonyi: The use of disk space is particularly frightening. We now have up to 1 Gbyte of disk space per computer, and all you have in it is copies of the same things, in the wrong versions. There might be a backlash in consequence of the kind of unorganised distributed systems that are emerging spontaneously. They have a lot of problems that are to the detriment of the individual. For example, upgrading to a new version of the software product is a major organizational problem. If it were centralised it would be a minor issue.

Hervé Gallaire: There are solutions if you do an upgrade overnight. People do not do it, but you can.

Gilles Kahn: Distributed computing is not necessarily synonymous to an anarchy.

Brian Randell: A lot of the systems that we have now have grown up because of UDI - unilateral declarations of independence from central administrations and services - by a lot of users.

Gilles Kahn: There are two kinds of systems, those that emerged from centralizing computing which got decentralised, and those that emerged from personal computers that got connected.

Charles Simonyi: The real question is whether the tendency will be toward creating a system that looks like the first type but is actually built like the second, that is, built incrementally. The first one was done by break down of the centralised system, not a build up. I think the build up approach has tremendous advantages in terms of growing organizations and scaling organizations. The break down has a lot of benefits in terms of the discipline that was brought, maintained or inherited over the break down. I wonder if we can achieve a synthesis of the two where you have the benefits of the incremental building up, and the discipline that existed in the centralised systems.

Take disk back-up for example. Often user machines are just sitting there, incredibly exposed. It is true that users are independent of the central machine breaking down but if their own system breaks down they are dead. It is easy to make the central system redundant - twice, three, ten times redundant, but two thousand times redundancy - that's nonsense.

Tony Temple: In my own case, much of the information is shared, and all the shared information is backed up, automatically. We all run agents on our own personal machines and every night the agents do the back up.

Gilles Kahn: In my organization 300 workstations are all backed up automatically.

Hervé Gallaire: This may be a difference between the UNIX and the PC world.

Tony Temple: We are talking about DOS, OS/2, UNIX, whatever.

Charles Simonyi: Tony, your organization is unusual - I take my hat off to you. My information says that the world is not like that. This is a legacy from the mainframe computer company - this is the positive legacy that you guys are not selling very well.

Tony Temple: It's not being sold through the normal retail outlets, but it's going into a lot of companies in one way or another. It's not a Shrinkwrap scenario - if it's infrastructure stuff it doesn't come from shrink-wrap.

Gilles Kahn: I want to bring up another problem related to network developments, namely that with multi-media applications there is more need for applications to understand how the network is performing so as to adjust their behaviour accordingly.

Brian Randell: No, you have to hide that at some level, in the same way that you hide issues of main memory and disk from application programs. Thus the abstraction that application programmers have of the network and what it will do for them needs to be more sophisticated than it was previously.

Tony Temple: Another issue is that when you go from addressing physical locations to addressing individuals, this is a huge transition. It will also be interesting to see how many commercial products will be addressable, as probably they should be, such as video equipment. This equipment is wireless-connected with no selection built in, but this can easily happen because of its shrinking so much. Once you have done this you can start to do new things that we never thought before. It becomes simply an addressing issue.

Charles Simonyi: Addressing is as important as connectivity. Pointing is a legitimate way of addressing things, just like in a GUI interface where you use the pointer to address files under a file name, in ubiquitous computing you will use your "pocket card" to click the various objects in the world, as opposed to having names for them. As a trivial example, if you want to identify a suit in a department store, you can use an infra-red device and simply point at it. Essentially, you can view the world as your desk top, and your infra-red device or smart card as a mouse.

Tony Temple: Many products have serial numbers, however they are not in a machine readable format and, in general, they are not unique.

Charles Simonyi: We have started on this notion of "plug-and-play" to put some means of identification into hardware devices, and now we are trying to take the leadership to tell the hardware people that it will also be for their benefit.

Bill O'Riordan: This will cause a "paradigm shift" in terms of how systems should be structured.

Brian Randell: The ability of connecting together fairly complex systems that were not connected before is an ability to make yet more complex, and if you are unlucky, uncontrolled systems, that is, systems whose possible failures you have not been able to predict, and systems whose good consequences you also are not able to predict.

Charles Simonyi: Will such systems be self-organizing with the notion of life of their own?

Mario Tokoro: Our life is more dependent on communications and computers, for example without communications and without computers we cannot live.

Tony Temple: The necessity is to have global real-time access to people and resources.

4.3.2. Need for new abstractions

Tony Temple: We have got another challenge, which is that main memory will be so large that to manage what is in the main memory will be as hard as to manage what is in large disks today. Disk management is already is a problem and main memory will be the same problem. We'll have 1 Gbyte of main memory.

Gilles Kahn: Another big change is that memory capacities now allow distributed databases where the local databases reside in main memory.

Brian Randell: A lot of the programming structures that we have at the moment are essentially legacies of the fact that we have main memories and these nasty disks, and we have built up a world of databases and programming languages and all sorts of dichotomies that no longer make very good sense.

Gilles Kahn: There is a danger that outmoded ideas about old technology will hold back progress. When a system contains a network cache large enough to hold a large database, the distinction between disk, memory and network should start to disappear.

Brian Randell: With sufficient changes in scale, even size of things, distinctions which did have valid visibility in the abstractions that people were using become invalid. There is always a legacy of abstractions that are no longer sufficiently abstract.

4.4. Software structuring

Les Belady: System complexity encompasses both people who must now work together in large, often, distributed and usually interdisciplinary project teams - and the software itself. It is this latter complexity that of software - that I now address.

Because these integrated, networked application systems must both offer a wide variety of services and be tailored to the particular needs of those using them, we see a split in the domain of software. I talk about this split in terms of Type A and Type B software. Type A is software that is, essentially, purely some component of the new system: an individual application, an off-the-shelf subroutine, a compiler, whatever. Typically, this type of software can be created anywhere, and is not particularly dependent on the specific domain in which the system will be used.

Type B software, on the other hand is the glue, the system's traffic controller, the software that holds the components together and integrates them into a smoothly functioning, domain-specific system. For instance, large Type A applications: computer-aided design, computer-aided manufacturing, inventory control, and so on, may be integrated by Type B software into a large enterprise-wide network. Usually created jointly by the software engineer and the customer, Type B software requires an understanding of the application area that only domain specialists can provide. Unlike its Type A counterpart, it cannot be created "offshore": in effect, this is the software that constitutes the fundamental nature of the system itself.

Brian Randell: Issues of software structuring can be divided into the problems of how to structure systems out of components and how to choose, identify or find components. There is a need for continued progress in techniques for creating structures. For many years I have regarded the issue of structuring as absolutely central. What makes one structure more effective than others depends on a number of characteristics. Agreement that some things are no longer important can be hidden behind a structure, for example we do not need to argue how many wheels a car has got. Apart from hiding things, a structure decides what can be thought about separately from other things.

Gilles Kahn: Concerning software structuring there are two different things, the first is the principles and the way in which we can connect things, and the second concerns what are the bricks that exist .

Brian Randell: There are several analogies that can be used, and each of these capture some aspects of what we talking about. There is the bricks and the mortar analogy, where the glue has an independent existence. But there is also the Lego analogy, where the glue is constituted by the ability to fit things together via a standard interface - both analogies have limitations, such as the static nature of the methods of interaction portrayed.

4.4.1. Glue

Brian Randell: UNIX pipes and filters, or object-oriented programming for that matter, really just provide a convenient form of mortar, with which the programmer can glue together whatever bricks he or she chooses to invent or obtain. But a new form of mortar does not help one to invent an improved set of bricks - the subdivision task (equivalently, the task of choosing interfaces, or of inventing languages) remains and still requires creativity (or at least an ability to recognise that some pre-existing component would suffice).

Gilles Kahn: About gluing languages, this is a research problem that is not well solved because people have all sorts of kinds of things to assemble together. In my opinion, we do not have enough ways of reasoning about the assembly process.

Brian Randell: So-called systems integration often means gluing things together which were not designed to be glued together - though there is a better word than integration, from old RAF slang, namely "to graunch", which means "to make to fit by the use of excessive force".

In recent years, a lot of people and companies have been proclaiming themselves as system integrators. This is because the past is, especially in the case of software, an increasing source of impediments to change, if not to progress. Such impediments have in recent years being engagingly termed "legacy systems". Most serious software cannot be designed for a "green field site". It has to support old interfaces, old data formats, old methods of working, and so on, as well as provide some improvements. Only by such means can it fit in alongside existing investments in hardware, data, software and people, and produce an overall economic benefit.

The ability to go forward and make progress despite the legacy of the past, despite what everybody else is doing, is exactly what real engineers have to do. But by no means all of our skills and tools are well up to the problems involved.

Logically, one might argue that legacy systems are by definition an ever-growing impediment, and hence will slow down development more and more. But who knows what novel techniques for coping with such systems might be developed, and prove so effective that the millstone proves ineffective. The very invisibility of software is a great aid here. One can glue ill-fitting systems and components together by a Byzantine network of emulators, converters, filters, etc., and rely on computer power to reduce their performance impact to a level which is acceptable.

Tony Temple: Legacy systems will continue to be supported in traditional organizations. These systems may be too big and complex to amend rapidly, or there may be no current justification for re-engineering them. Client/Server front-ends for these legacy systems is an emerging software market.

Hervé Gallaire: You will see middleware platforms made more widely available to solve one aspect of the problem of legacy systems, and you will see better gluing languages to combine already made applications.

Bill O'Riordan: There is a slight problem with integration. The retailers and banks are now doing their own integration, and not using any integration tools. The other thing is that the way they integrate and build their system is their differentiation.

Hervé Gallaire: As soon as the customer will understand that there is something on the shelf which does this minimal integration, they will not pay the integrator to do it. The integrator will have to do, maybe, a very specific part which is specific to the business.

Gilles Kahn: Integrators will have problems because all components have a life on their own which keeps changing; the integrators will have to adapt to the components' evolution. If it is not mechanised the integration become cost ineffective.

Brian Randell: Integrators are advertising openness but selling closedness.

Gilles Kahn: You can play that game, but I am not sure if you going to win it. I guarantee that the attitude and practice of having software which is available from only one manufacturer will not succeed.

Bill O'Riordan: Different software gives you the differentiation, and if you want to compete with someone you better have something that is totally different.

Brian Randell: What Gilles Kahn means is how do I plan for no longer using such unique software? What is my escape route? This has to do with exportability and importability. Is this software taking me down a cul-de-sac? Are the advantages of going down the cul-de-sac worth the risk involved? Increasingly, the answer has been negative.

Hervé Gallaire: We move from a world where the management information systems (MIS) shop was developing applications and delivering them to the end-users, to a world where end-users become power users; they want to move away from the integrated application, and tailor the suite of applications to their taste, to the requirements to get the job done quickly, to do it in reactive ways, depending on the situation in the work group at a given time, depending on the availability of resources on the network, and so on. Current programming languages are not languages that are suited to this kind of task, as evidenced by the proliferation of dedicated scripting languages, such as Apple Script, Tool Talk, Visual Basic and others. Although they provide meaningful abstractions to the user, it is clear that the way these languages have been defined only takes into account simple ways of connecting modules on the network, with little abstraction power and no reasoning capability. It is clear that the requirements will become more stringent, as the usage of these languages become more widespread and move to the end user. What is needed then, is the development of true "glueware" tools, providing the level of mechanisms to deal, beyond the obvious parameter passing, with for example synchronization, co-ordination, identification, localization and failure analysis requirements.

4.4.2. Components and reuse

Brian Randell: What we have got is that, over the years, we have developed better forms of mortar, better forms of gluing things together, and gradually we found useful sorts of bricks, some of them very limited but within their domain very valuable, and others which are rather more general. The most general purpose types of components that I know of are ones which have no effect on the functionality of a system, just on the way it delivers this functionality. For example they provide you with a facility to program on several computers as though you are on just one; if you have a component that does that, you can apply it to a lot of different applications. There is another class of component, namely those that provide a function of general utility. An obvious example of such facility is a GUI which hides a lot of programming problems related with the use of windows. Such components are equally applicable across a very broad range of application areas. Incremental progress has been made by people having bright ideas which help identify such bricks and by providing good implementations of them. However this has been done in a way which is very anarchic.

I will draw an analogy to the hardware design world I saw in IBM back in the 1960's. Designers would struggle to decide when a new hardware module had to be designed, and whether a newly-designed component was worthy of incorporating in the official library of standard components. The design of a system, say a new CPU, would be criticised if it involved designing too many non-standard components. There was a great prestige involved in getting the interface and designing of a new module right, so that people agreed that it was worthy of being placed in the library. There was a brief period when this discipline seemed to work very well; perhaps it still works much better in hardware than it does in software. There was a corporate-wide discipline of controlling the number of different components that you had. This discipline was underpinned by the strong limits that the physical world placed on number of things you could possibly put in a single component.

It is the lack of an equivalent sort of control that it makes very difficult to predict what is going to happen with regard to standardization of software components. Often Microsoft, for example, designs or adopts a component which consequently suddenly becomes the standard that people have to use. Other times industry organizations, like the OSFs and OMGs of this world, have a bigger effect on decisions related to inventing, selecting and settling upon general-purpose components.

Charles Simonyi: I agree with you that such components or bricks, in other words shared artefacts of some sort, are what is desired. However to think of such artefacts as being simply OOP-style objects or even subroutines is a mistake. Actually, I do not like to use the word object because of the specific connotation of inheritance. When we have talked about objects we really talked about shareable artefacts, and components.

Mario Tokoro: It might take years to understand how to program well in terms of objects, then we may need something in a higher level of abstraction, like what I call autonomous agent. In fact an agent will be composed of concurrent objects, in much the same way that a person is composed of cells living concurrently.

Brian Randell: This is pretty much like to the original notions in SIMULA which got rather lost.

Mario Tokoro: The difference is that, at that time, in the 1960's, people would write their own single piece of software, but nowadays I might use the software that you have written. Nowadays programs are made by many different vendors so that users can choose those that best serve their needs and budget. Software vendors use other software vendors' software as their components. Up to this point users have a physical copy of the software on their machines.

However, the next step is a distributed version. Here the users don't have a copy on their machine. Instead, they only have a calling program and the right to access programs on the vendors' machines. In turn the software on the vendors' machines will call program modules residing on machines at other software houses.

Hervé Gallaire: This is a form of gluing which takes into account the concept of servers.

Mario Tokoro: Another thing is that we have been thinking about static complexity of software: How do you combine? How do you optimise? How do you reuse? But when we start to think of the software of the entire system composed of individuals (autonomous agents) then it is a complex dynamic system. So far we do not know how to deal with this type of dynamic system. Static complex systems can be a collection of parts connected together for a particular level of functionality. Dynamic complex systems are not like that, the collection of some parts show some different behaviour than just the collection of the behaviours of the parts.

Gilles Kahn: How do you see object-oriented programming? As a technology which lasts for a while until it is replaced for something else, or as something which is intrinsic? My current view is that OOP will be subsumed by a more static discussion of dynamic behaviours. In physics all the dynamics are in the equations which are extremely static things that we can master. We have to think for the moment of the OOP style as a temporary measure because we are not still capable of representing a computation in a static way.

Hervé Gallaire: I think there is no disagreement between what Mario Tokoro and Gilles Kahn have said. The object may be the static part, but still you have to have the behaviour part which may be a set of equations or a set of constraints.

Charles Simonyi: I think we should not use the same word (object) for the two meanings which are being attached to it, which are completely different. The objects we talking about are software engineering bricks which are very useful and will not disappear, just like structured programming which has not disappeared, rather it has been swamped by new concerns.

Brian Randell: The point raised by Mario Tokoro is that it is crucial to deal properly with the problem of making controlled changes to systems which are in continuing operation. One of the major current software engineering challenges concerns how to make changes without stopping the world. Others are: how best to breakdown large tasks into manageable subtasks? How to make use of components that already exist? How to try to design components so they can have multiple uses? We have improving technologies for putting bricks together, but the notion that the whole industry is going to move forward and gain the sort of reuse that people claim they will get, is one that I am sceptical about.

Tony Temple: A few companies are now emerging who specialise in developing or buying software objects, and who will sell the resultant object libraries to integrators.

Charles Simonyi: We think software reuse is critically important - this is hardly controversial. Our experience has been that reuse has remained superficial even when:

- "not invented here" has been eliminated (by moving personnel),
- incentives have been employed (for example, by stock options).
- reuse opportunities have been identified (by technically savvy management), and

- object-oriented techniques have been employed (using C++).

What remains after all this is an inherent mismatch of goals between the producer and the consumer of a shared artefact. The producer wants it to be general so that the market will be sufficiently large to support the effort, while the consumer wants something highly specific - so that the performance and feature set will he comparable to what is needed and what could be produced separately. The fact that the producer and the consumer are within the same organizational umbrella does not really change the picture under the following conditions that are normal in our industry:

- the shareable artefacts are relatively small - the larger the artefact the larger the variance in the requirements - often non linearly larger, (for example, due to the combinations of subsystems implementation possibilities). Even for small artefacts, the requirements are seldom if ever identical (for example, in the implementation details of parameters).
- general solutions (solutions lacking in domain specific knowledge) perform worse and often non linearly worse than special solutions.

Communication costs between programmers are very high (especially with the smaller artefact) and can be non linearly high if trade-offs have to be negotiated.

Charles Simonyi: I am positive that greatly improved levels of component reuse are coming. This will be when we create new abstraction mechanisms that are combined with specialization mechanisms at the same time. Every time you abstract something you can also have a mechanism to specialise it, either to exploit domain specific knowledge or to obtain performance. The use of subroutines is an example that always goes in one direction, you always abstract, you cannot take a subroutine and specialise it.

Hervé Gallaire: We do not understand reuse well enough; we expect too often total reuse from a reuse approach, independently of the context in which it appears.

Brian Randell: The question is how much reuse you actually are going to get, which might depend on a set of issues which are essentially non-technical, which might be organizational or due to the market place.

Charles Simonyi: Reuse did not win in the past essentially due to technical obstacles. Once the technical obstacles are removed, by employing for example the meta-object protocol and similar specialization mechanisms, it will be natural, like opening the floodgates. Does this make sense?

Bill O'Riordan: It makes sense but it will never happen.

Charles Simonyi: It never happened because we were always generalizing and we were not specializing. Every time you added a generalization to something, you added to the cost of reuse.

Hervé Gallaire: Programming will be put in the hands of different people. Until now, people were very technical and never liked to reuse. If you are trained to just get a job done, and not to be a programmer, then you will reuse.

Tony Temple: It will increase, and I am sure of that but, if you want to have a lot of reuse, you have got to know what things you can reuse. If you are less expert, you will know less about what you can reuse. Apart from that, there will always be cases where people will say that this brick does not do what I want it to do.

Charles Simonyi: That is because we cannot make the bricks that are general enough. Every time you make a brick more general, it gets less efficient. But if you combine the generality with specialization then you will not have the problem.

Bill O'Riordan: When people buy a database, for example, they modify that database to satisfy their business requirements, often with very naive programmers. Once they start to modify it, the database becomes non-reusable, and becomes a legacy problem the next time we try to upgrade.

Charles Simonyi: Reuse is broken by making modifications, and people make modifications because flexibility has not been presented in the artefact. The reason for this is that putting too much flexibility in the artefact, using today's abstraction capabilities, makes the artefact perform so poorly as to be useless.

Brian Randell: Reuse in terms of components currently works to a degree, but I believe the number of distinct objects will grow to form a very large library, and I am sceptical as to whether it will be easy for people to find out whether there is a object, or brick, of appropriate utility to them.

Charles Simonyi: Your expectations is that the number of objects will grow radically, my argument is just the opposite; the number of objects is not going to grow. What is going to happen is that the objects will become infinitely more flexible because flexibility will not influence run time costs.

Charles Simonyi: I will give an example. In the McIlroy speech at the 1968 NATO Conference there was this notion of bricks, and a prediction that 300 sine routines might be necessary. What I am saying is that it should be necessary to have only one sine routine, or perhaps just one trigonometric routine, but with 30 or 40 arguments. One of the arguments will be the number of bits of precision, another argument will be the required speed, and another will be the type of trigonometric routine. All the things you can think of will be arguments, and what you will get is a partially evaluated copy of that infinitely general trigonometric routine for you to use.

Tony Temple: I see the reverse. People are trying to get simple parameterization of things, so they fragment problems, and also for performances reasons and less code. And this trend will be in all areas, from object-oriented objects down to simple things like trigonometric functions.

Gilles Kahn: What will decide between these two approaches will be a commercial issue, the market will decide.

Hervé Gallaire: Whoever has been involved in the design of large pieces of code, or of libraries, in the most modern object-oriented language, will undoubtedly have experienced the difficulty of designing for reuse; constant trade-offs have to be made to achieve this goal, but these trade-offs are not apparent any more to the users of the objects to be reused. We believe that there is now evidence, from our work and at other places, that it is possible to design "multi-ported abstractions" that honour the concept of abstraction, and yet do this in ways that makes it possible to later go back and, through a parallel abstraction, address some of the previously hidden issues. This ties in with what Charles said about specializations and corresponds to the technique we use of Meta-Object Protocols.

Gilles Kahn: In hardware, generic components are becoming much more common.

Bill O'Riordan: We have for example the "intelligent PAD" which is dynamically reconfigurable hardware which optimises itself - every time it runs you get better performance. It's cheap and it uses standard parts.

Gilles Kahn: We keep getting asked why don't you use components like the people in hardware.

Charles Simonyi: This notion that software should be like a semiconductor catalogue has been misleading. This is the most common view that people have about reuse, and people always pursued reuse in those terms. When hardware designers start reaching the levels of complexity where we were in software twenty years ago, they will have the same problems.

Brian Randell: When you next find a new microprocessor which comes onto the market without a design flaw in it then you can start to believe such comments.

4.4.3. Beyond objects

Brian Randell: Present notions of object are better than we had before, but are not what we will need in the future.

Charles Simonyi: OOP is not the complete answer because we do not have a static view of the picture yet. You can make a comparison between OOP and structured programming. At the time, structured programming was over hyped, but today nobody would dream of branching into the middle of a "while" loop, so the ideas have lasted.

Mario Tokoro: We need an evolved notion of objects, which we call autonomous agents, to describe open and distributed systems. An autonomous agent is the unit of individual software that interfaces with humans, other agents, and the real world in real-time. Each autonomous agent has its own goal, and reacts to stimuli, based on its situation. It behaves so as to survive. A collection of such autonomous agents shows emergent properties which cannot be ascribed to individuals, eventually forming a society.

Charles Simonyi: A great frustration is to work on a new program, and not be able to use old ideas. Usually our software artefacts become totally obsolete; such artefacts should be independent of hardware and language. The thing we should try to capture is computational intent instead of implementation details. The moment you can do that, you are well on the way to making at least one part of your software immortal, namely the computational intent.

As an example, the computational intent might be to sort a collection, where the implementation of sorting a collection might be to do nothing if the collection is implemented as a sorted list. This is not the same idea as a subroutine since a subroutine implies a particular implementation. I am frustrated at having had to encode ideas that I feel are still applicable today in some language that is now obsolete. OOP does not fully encompass this idea, although it is a step in the right direction.

Herve Gallaire: It is possible to provide powerful connections between computation and declarative modelling, by developing programs as sets of constraints, through the idea that computation is the processing of specialised systems of partial information. We have coined "model based computing", an approach in which we rest upon specialised constraint systems, in virtually every arena of computation; we are experimenting with this approach for applications in embedded real-time software. What is most appealing, in the true engineering tradition, is that it becomes possible to build the software so as to have an implementation, but also to allow monitoring, simulation, control, diagnosis, explanations of the system so developed.

Gilles Kahn: It is clear to me that, if we had the intent of the computation you could help the developer within a given context to reach that intent, and you cannot if you do not know it. In symbolic computation, we cannot help at all before it is explained what has to be done, as opposed to the desk calculator where you begin to push keys before explaining what you want.

Brian Randell: This notion of "intent" can be interpreted, perhaps misinterpreted, as simply specification. It is sometimes argued that if we have adequate means of specifying things then to a large extent programming will go away, since the transformation of a specification into some adequate program could be semi or fully automated. My view however is that such specification is in some senses just another form of programming and in some cases it has more difficulties than the actual programming.

Charles Simonyi: You are absolutely right, and I am not going to disagree with that. My purpose was not to simplify programming at all, my purpose was to factor programming to components that would facilitate reuse, and to facilitate immortality. It might make programming more difficult, and that would be all right if immortality results in a higher degree of reuse.

Charles Simonyi: Intentional programming addresses each of the following phenomena:

- implementation detail is separated from computational intent (hence the name "Intentional" programming) so we have the benefits of program generators but without their costs; variances in requirements become parameterised invariance.
- domain specific knowledge can be added routinely using program transformations.
- trade-offs are simply eliminated by providing great abstraction capabilities (to express the commonality) and similarly powerful specialization capabilities (to express the specific needs without run-time costs.) The latter is achieved using partial evaluation of the general code.

It is also important to note that under intentional programming, the concept of shareable artefact includes not only subroutines but also transformations which would be typically performed manually by programmers in each instance. For example a storage allocator library routine is a normal subroutine in C. However, the act of updating a program so that an array is moved from static storage into allocated storage is not typically considered an artefact, it is just work that programmers do. Under IP, the updates (transformations) would be part of the package together with the allocation subroutines. This means that we not only want to improve the chances of reuse for any given artefact, we also want to make more of the programmer's contribution into potentially reusable artefacts.

The basic system does not involve any artificial intelligence. Our goal is not to simplify the "thinking" part of the programmer's job, but to help the programmer with the more mechanical aspects of programming. For example, the programmer has to make the decision about what implementation method should be used, or even whether the pre- conditions for a restricted implementation exist or not. The system will, in turn, apply the transformations that are required. This has been called by some "semi-automatic programming". The programmer's work will not get simpler, in fact programmers will have to think more per hour. In return the work will be much more reusable.

Gilles Kahn: What we are saying is that when we design such an object we do so without a priori knowledge of the context it will be used in.

Charles Simonyi: And it has abilities to adapt to its context.

Mario Tokoro: I see objects as components in an organism. Each object serves the complete body, or agent. The glue is the interface between agents. These autonomous agents need to incorporate some notion of environmental awareness. There are new problems surrounding this type of agent. They are the same types of problem that surround parallelism. Traditional systems are essentially static and problems can be solved by fragmenting the system into components (divide and conquer). This is the traditional approach. Distributed systems are dynamic and cannot be solved in the same way.

Charles Simonyi: When we speak about autonomous environment-aware objects we associate two concepts to them: "desire" as described by Mario is what the object wants, and "intention" is what the programmer wants, which will be codified. When an object is put in the system the intention gets expressed, objects carry the programmer's intentions.

4.5. Software engineering issues

Les Belady: Even today, for many people, software engineering is essentially a management discipline. Today even this model of software engineering is no longer adequate; it will no longer work. Why? Because of the nature of recently appearing new application systems with an added dimension of complexity: communications and networking. This leads to new kinds of programs:

We have seen that this world is undergoing a shift in mission that will utterly transform the software engineering discipline. Because integration is the key to creating large, complex computer systems, software engineers must become application system designers, and ones with significant hardware and application domain expertise at that. These designers must take, an often staggering number of components, many of which are themselves quite large, and combine and recombine them into the integrated networked super-applications of tomorrow.

4.5.1. The state of software engineering

Brian Randell: Despite all the software research and development that has been undertaken, the two most important events in the history of software since the invention of the stored program concept were the development of (i) FORTRAN, and (ii) the microcomputer! (The former established the recognition that it was practicable to program at a level which was significantly more abstract than the machine code level; the latter caused what was in effect a huge programmer and user population explosion.

Hervé Gallaire: Design methodologies are poor, in general: OO design, client/server design, distributed or parallel system design, user interface design, real-time design are examples of domains in which we have very little at hand. We do not understand well the analytical aspects of software engineering, even when it's possible to develop them; for example, we do not really understand the performance issues of various database techniques, or of various networking schemes, even though this should be possible. There is a lack of real standards, even in well understood areas such as SQL and C++.

Brian Randell: The term software engineering was in fact introduced in 1968 to describe a requirement rather than a reality. One can argue that a number of the established engineering disciplines, as they become involved with ever-more complicated designs, are becoming enmeshed with the sort of problems as us, albeit ones that we have not yet solved very well, such as predictability of the design process, and so on. Alternatively, one can take the critical view that there are a lot of things done by some of the more established engineering disciplines which we would do well to mimic better.

Gilles Kahn: My personal opinion is that software engineering has been progressing slowly and systematically, rather than dramatically. But the need for better trained people who work more systematically is more and more recognised, and there are still huge areas where the people involved have had no training at all in computing, and therefore produce quite substandard results. So even though it seems there is not a very good case in the industry, I still think that in industry at large substantial progress is taking place, and programmers are getting better and better.

Hervé Gallaire: Why are programmers better? Is it because individuals are better or because tools, environments, and software are better?

Charles Simonyi: Education is longer, the teachers are better, the curricula concentrate more on computing science rather than other engineering disciplines, and I think that many of the arguments that usually consume a lot of time, like those about structured programming, have settled down.

Gilles Kahn: In many systems, software has become the dominant part of the system cost, for example in military weapon systems and in large telecommunication systems. In trying to understand how to build the next generation of telephone switching systems the argument in favour of having more systematic software development is no longer challenged. Also the fact that "there is no silver bullet" is slowly becoming more acknowledged. Is my picture a bit rosy?

Brian Randell: In military systems there is a tendency for the hardware manufacturers, for example of sensors and of weapons, to produce their individual subsystems and then make a belated attempt to use software to glue everything together, rather than have a view of the whole thing as a system, and have a suitable overall system architecture in which the software architecture plays its proper role. However, software organizations are becoming more influential with respect to the overall design of the system, though in some situations they are just left with the task of brushing the overall complexity under the carpet, so to speak.

Gilles Kahn: Sometimes I wonder whether computing is not taking the blame just because a scapegoat is needed.

Charles Simonyi: Sure, specially in the American newspapers where the term "computer error" is often used instead of "human error" or "errors of interaction with the computer".

Brian Randell: Software can be used to encode solutions to very complex problems more cost effectively, and faster, than in any other technologies. Almost by definition, therefore, it is where the complexity that results from overambitious goals ends up, and where in consequence one creates somewhat of a history of large and unsuccessful systems.

Charles Simonyi: From my view from the PC world I saw the opposite. It was always the small innovators who surprised the world by going beyond what people thought was possible. So I see it more as underambition rather than overambition.

Brian Randell: The question is whether there is a difference between software engineering and the other disciplines, in terms of the amount, frequency, and seriousness of projects which fail essentially through overambition of the application.

Tony Temple: Take the Channel tunnel. It took longer, is more costly, and so on, than expected. We have failures in all the fields.

Bill O'Riordan: Projects that failed due to overambition have been mismanaged because they should have been cancelled before the situation reached that point.

Gilles Kahn: In general this overambition does not necessarily come from the provider of the software, but from the user.

Brian Randell: The history of engineering teaches that success leads to failure, and failure leads to success. One of the problems of software engineering, compared to other types of engineering, is the increasing amount of innovation and complexity that you get in a series of successive projects. I would suggest that the sequence of bridges built by a bridge building firm will typically involve much less design innovation, from bridge to bridge, than the typical sequence of projects designed by a software house. This perhaps is less so now in the software package world where you have, for example, Word 1, Word 2, and so on, and in that sense it is becoming more like ordinary engineering.

Gilles Kahn: Are the failures in those projects due to technology reasons, such as the fact that the people are using a technology that does not scale up?

Brian Randell: The sort of failures I had in mind are related to situations where there was an inadequate understanding of what would be a sensible overall system architecture, and of the overall complexity of what was being attempted. Looked at sufficiently abstractly the system looked simple, but there were so many odd things that there was a level of combinatorial complexity that had not been appreciated at the outset.

Brian Gladman: Software engineering still has some way to go in design understanding - predictive design is still fraught with difficulty and many large systems (the handling of which is a good test of maturity in itself) still go badly wrong. However, in some areas (compiler writing, database design and so on), there are now good models so we are clearly on the right road to success. Are we about 50% of the maturity curve?

Another good sign of maturity is the emergence of component-based systems construction and a supporting industry infrastructure (that is a component supply industry). A fully mature discipline will have several industry supply "layers" and there will be very good, long standing standards, especially simple ones (5 volts logic, the standard house brick, and so on) In the software world the position here is not clear - some standards have emerged (for example, languages, library interface, application portability interfaces and so on.) But it is not clear that there is a true industry infrastructure which supports component-based construction. Some company specialization is now occurring and there may be the signs of emerging component supply companies with the development of C++ but it is very tentative. On this basis, I suspect that we are low on the maturity curve.

Brian Randell: Has the idea of software engineering been a success? If so, what can we cite as evidence of this success. For example:

- There is now a computer science department in most universities!
- The productivity and expectations of programmers has gone up enormously
- The comparative ease with which all sorts of applications are put together because of the success there has been in creating components of general utility.

Charles Simonyi: Look for example at operating systems such as Windows - you can regard the operating system as a very large shared graphics library, which really wasn't the case before. But this is a library which is shared by every program.

As a result of the ensuing discussion, the Technology Working Group produced the following non-exhaustive list of significant recently developed, or recently successful, software innovations, in order to try to indicate the amount of progress that has been in fact been achieved in software engineering:

- Object oriented programming (OOP) which integrates the notions of encapsulation, polymorphism and inheritance.
- Remote procedure call (RPC) and what it has enabled, for example: client/server architectures.
- Application programming interfaces (APIs)
- Effective ad-hoc standards, for example: Standard Query Language (SQL).
- High level languages applied to modelling, for example, logic programming and constraint programming.
- Program transformation.
- Source level debugging.
- Multiple instruction multiple data (MIMD) in parallel computing.
- Graphical user interfaces (GUI) and GUI development tools.
- Advances in decompression and compression.
- Emulation improvements.

4.5.2. Tools

Bill O'Riordan: With well-designed tools, even developers need not know anything about the memory architecture of the system they are using, any more than C programmers need to know the assembly language of their CPU. The end user is even further removed from these concerns. Now that good tools and operating systems are available and machine-independent applications are finally being delivered, users are getting their chance to compare products based on criteria that really matter: what applications are available, and how do they perform?

Gilles Kahn: The cost of the tool is one thing, and the cost of learning this tool and adapting it to our way of working is another, which is usually dominant.

Hervé Gallaire: Tools are not portable, they are at the moment too tied to specific platforms. The situation is still a mess; PCs have certain type of compilers and workstations have different compilers.

Tony Temple: Progress has been immense, certainly no worse than anywhere else. There are different classes of tools; in some places one tends to find bad tools, and in other places one might find excellent tools. The problem is that today's IT user tends to work on a variety of different environments and platforms (hardware and software). The only way to get around this situation is to have a common front-end. We have a desktop which will allow us to run both Windows and UNIX, for instance, and have libraries which increase code-sharing across the whole system; we can thus run any tool we like in any system we like, in one desktop.

Brian Randell: Software tools are just another application area, and comparing this application area with some other application areas we can see that there is considerable scope for more improvement, market shake out, transportability and so on. Another issue is that there exists a big distinction between generally available tools on the market and the tools available in a number of specific areas and environments.

Gilles Kahn: Another topic that we should discuss is related with the issue of integrated tools versus a multiplicity of completely disconnected tools, available on the market without any notion of integration.

Hervé Gallaire: What about the portable common tool environment (PCTE)? This is different from CASE tools in the sense that it is more like a software bus, into which you plug your different tools. It is an environment where a compiler shares objects with loaders, and so on. It is becoming standardised. Is this important?

Bill O'Riordan: We have walked away from PCTE as a company.

Charles Simonyi: I am convinced that integrated systems will be the future systems, but you need a tremendous capital investment for integrated systems, and only very few manufacturers can make that capital investment.

Gilles Kahn: Here there is something which is a little worrisome, namely that the software supplier is sometimes required to follow a particular method to design and implement a given product. To a certain extent this is understandable but it is a way of stopping certain kinds of competition.

Charles Simonyi: The supplier should be pro-active and offer a greater menu of possible choices of methods, or offer a better description of what they are doing so that it becomes acceptable to the customer.

Brian Randell: It is also an issue of relative customer-supplier strength. For example, the set of customers for real-time systems in the UK have a set of rules about the process and tools which their supplier must use to develop software. They have got strength, a buying power, which cannot be resisted by the software companies, which are comparatively small.

4.5.3. Project organization

Brian Randell: Now and in the future, we will need people from different organizational units and different locations to work together. Has this problem been solved?

Bill O'Riordan: We are doing this at ICL with some success. Distributed people provide fluidity to close down projects, or redeploy people at short notice. There is a growing problem however with working from home. In this case, the person cannot escape from their work or home environment, and this can lead to sociological problems.

Charles Simonyi: We spend a lot of money to bring everyone together at a single site. We feel it is important for people to feel part of a team. When this happens, they have fun, take pride in their work and produce good results. You say that this becomes a problem when the project comes to an end, but inevitably some people will move on while others stay to start on the next version. Any ideas that are not included in the current version become the starting point for the next. There are benefits to reducing the size of the group, provided you can increase productivity.

Tony Temple: There is a tendency in Europe for programmers to work at home, at unusual times, during nights and weekends, which puts the requirement back to decentralised organization of groups.

Charles Simonyi: We are discovering that too. We have almost the same problem in our 25 buildings in Redmond.

Bill O'Riordan: We had to build a whole organization because we have the same problem: people want to work from their homes.

Brian Randell: There are a set of experiences and a set of problems that apply to teleworking, and it is very clear that software development is moving in the same direction.

Brian Randell: When one talks about co-operation, there is an important distinction between what one could describe as organised and planned co-operation and accidental co-operation. A lot of accidental co-operation occurs if there is accidental communication, such as occurs through use of the same common room. There are a couple of attempts at getting that sort of co-operation across a network between geographically separate sites, such as the Xerox PARC scheme of joining the coffee lounges of two laboratories, each having a video wall showing the other coffee lounge.

Bill O'Riordan: Video conferencing is very interesting. This is not only because you get a very good face to face communication, it is the only way we can actually transmit diagrams and images, and discuss them quickly, because if we try to send them down the network it would take a lot of time.

Tony Temple: This is quite interesting because before we have a video conference we fax all the documents.

Les Belady: Co-operative work (CSCW) sprang from the awareness that while the first decades of computer use supported only the individual - with the machine used interpersonally only for purposes of communication, at best developments have evolved toward assisting people as they collaborate and work together. One recent thrust is groupware, which supports on-line collaboration via networked computers or workstations. Examples include the co-authorship of complex documents (such as system specifications) and such efforts as COLAB at Xerox Parc. Using groupware, teams can use the computer to collaborate across distances or, as in the case of shared work surfaces for conferencing, in the same room.

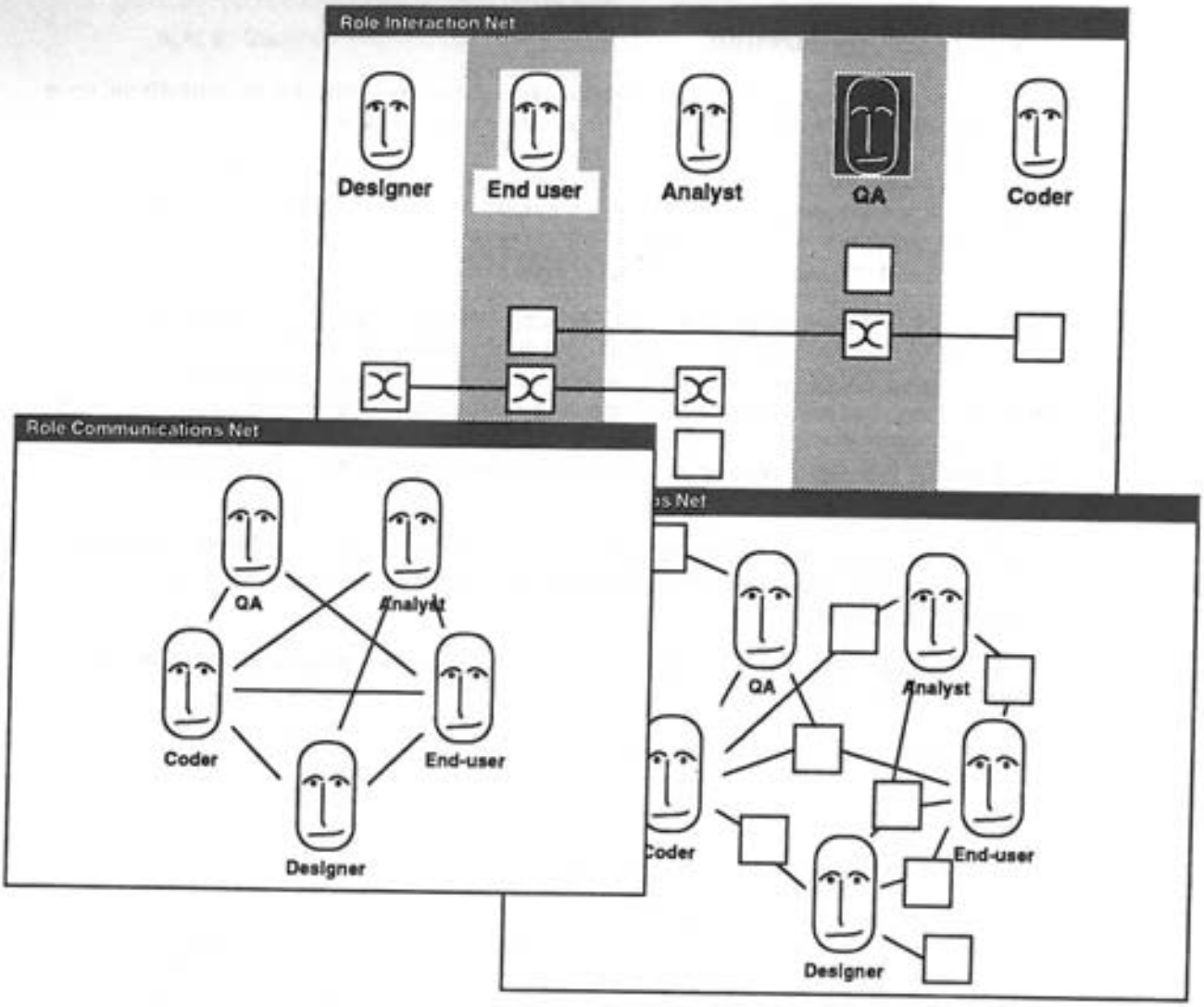

Figure 4.2. Users of MCC's Co-ordination Technology can capture and edit complex processes.

Groupware tools typically support direct, session-oriented collaborations that occur at discrete points in time. Another important area of CSCW research, Co-ordination, is somewhat different in that it supports large, complex projects over longer periods - often over months, or even years. Essentially, Co-ordination systems seek to computer assist multifaceted human activities, such as planning, designing, and implementing complex processes - ones which involve, for example, both people and machines. Many of these systems, such as the Co-ordination Technology system under development at MCC's Software Technology Program (see the above diagram), provide tools and methodologies for capturing existing processes and designing new ones, as well as for viewing and editing these evolving processes.

How might computer aided co-ordination of people and resources enhance the process of large system design? For one, studies have shown that the quality and productivity of the group depends more on interaction among people than on the sum of all individual actions. Co-ordination systems consider a project (or some other organised human activity) to be a distributed, asynchronous amalgamation of actors, roles, teams, actions, and interactions. The systems then order these elements so that people know when events are to occur, what an interaction between roles is meant to accomplish, and how interactions affect other interactions. Because scheduling and development are integrated at the workstation level, a project's management load is lessened, leaving more time for motivation and creativity. There is a better exploitation of concurrency, leading to a reduced "product" cycle, as well as better documentation of the process involved in a complex project. This latter advantage allows for greater transferability of accumulated experience, as well as better training of new staff via computer models of the project.

4.5.4. Process models

Brian Randell: Early models, such as the waterfall model, are being abandoned, as being too naive.

Charles Simonyi: The waterfall model does not apply in either small or big programs. The basic problem with the waterfall model is the lack of validation at each step of development. The issue of program validity is concentrated on the final step of development.

Tony Temple: Even the new methodologies are equally process oriented, in the waterfall sense. For example, the object oriented analysis (OOA) is at the end almost waterfall. Although you can get a better feedback validity of your specifications because they are in terms of objects, the waterfall model is still there in the subsequent steps of analysis.

As regards the role of prototypes in process models, one must be clear that there are two types of prototypes, one which is evolutionary and another that has to be thrown away once a particular stage of development is achieved.

Gilles Kahn: Together, with the prototype machinery, we have to set some means of going systematically from the prototype to the final program, however this does not have to be entirely automatic.

Les Belady: Systems Visualization is another knowledge engineering research area that holds great promise for the world of large system design, in that it increases insight into complex systems. Already, Visualization is a major feature of many scientific and engineering applications: scientific researchers have found that "what if" computer simulations, ones that allow the user to visualise hypothetical results, are often far more useful than concrete experimental equipment. An example is, of course, wind tunnel visualization.

In the context of software system building, we have a somewhat thornier problem, however: the problem of visualizing the non-tangible. What, for example, is the shape of software? Traditional flowcharts are not particularly helpful for quickly recognizing interesting characteristics. Perhaps standards for software shapes, like standards for windows, will help. Another problem is how to visually represent the dynamics of a system. Currently, the use of displays is little more useful than the traditional "sheet of paper" mode of System Visualization, and animation and artificial reality programs, however tantalizing, are still slow and cumbersome. These examples show, however, that a good amount of excellent research is underway in the area of Visualization, and I predict that results will soon have a tremendous impact in the world of large system design.

Alessandro Giacalone: Vision is the primary conduit of information into the human brain. In terms of support for human reasoning, decision making and action, visualization is one of the most useful, feasible and fundamental functions computer technology can provide.

4.5.5. Dependability

David Talbot: Dependability already confronts those providing software driven control systems of all types; from all forms of transportation systems to those concerned with industrial and environmental control. Here the driving forces are usually safety and other legislative pressures. However, dependability from an economic and competitive perspective faces those who are putting increasing bodies of software into "traditional" products such as washing machines, the electric shaver right up to the TV set which is now a major software driven distributed system in its own right and which, in turn, will become an entry point to the "information highway".

Peter Wharton: Advancing technology does not notably diminish the rate of failures in large complex projects; perhaps because our ambitions grow as fast as our capabilities. The three main reasons for project failures identified by Curtis et al (ACM Communications, 1988) - the thin spread of application domain knowledge, fluctuating and conflicting requirements, and communication and co-ordination difficulties - look set to remain unsolved for some time. If anything the rate of change of user requirements is accelerating.

Brian Randell: The various developments that we are talking about are pushing us headlong down a route towards huge complex systems which at all times contain parts which are not working. We tend to have in the software world the naive notion that something is either working perfectly or is not working. But the world as a whole is made of things which do not work perfectly. Perfection and correctness are mathematical concepts rather than engineering concepts.

Gilles Kahn: This situation is changing a little bit with networks because very often you are looking for a service but you do not know exactly from where the service will be provided because some of the computers may be down.

Brian Randell: With regard to dependability, the most fundamental characteristics of software are that it typically embodies immense complexity and it is intensely brittle. The brittleness is of course logical in nature, and comes from its digital character. No recourse can be made to normal notions of continuity and of safety factors in designing the software, in validating it, in estimating its dependability, and so on. Research on probabilistic algorithms, and on inherently self-checking programs, could be regarded as embodying a slight move away from this situation, but logical brittleness is likely to remain a major characteristic of software, and continue to impact our ways of producing and using it. I foresee at best only slow incremental improvement in our ability to deal with this characteristic.

The principle techniques for ensuring highly dependable software are testing, formal verification and fault tolerance. These are to be regarded as complementary - each has its limitations. Used together these limitations can be minimised. However, it is very difficult to determine how best to divide limited resources and time between these three activities.

Incidentally, a few months ago Tony Hoare came up to me and said that he had started to think about the sort of amazingly sophisticated software packages that you can buy now - he specifically mentioned WORD as an example. He said that though such systems were not entirely bug-free the amount of reliable functionality they provide is very impressive. He went on to say "it has nothing to do with recovery blocks, you know" but before I could respond he admitted it "is nothing to do with formal proving either".

Charles Simonyi: We make some use of fault tolerance, in addition to testing, but not formal verification.

Brian Randell: In the ESPRIT project that I lead there has been a lot of concentration on the problems of testing and validating software systems. We have concluded that there is little conceptual difference between software and hardware in regard to testing and validation. However, in practice there are issues of the feasibility of conducting enough tests, though the beta testing used in some parts of the industry is very effective.

Charles Simonyi: It is indeed very effective. If we consider the scope of beta testing for the Chicago system, we are talking of 10,000 people.

Brian Randell: Believable guarantees that a very high level of dependability (for example, 10^{-9}) has been achieved for very complex software are beyond the state of the art, whether one uses any or all of these techniques. Lower levels (for example, 10^{-4}) can often be achieved - even by applying only testing. The aim therefore must be to achieve a balance of dependability and dependence - avoiding unnecessary dependence. If cost-effective non-software protection or back up could be available, then we have to make sure that is employed.

> The figure 10^{-9} translates as "the probability of failure is 10 to the power minus 9". Similarly, for 10^{-4}.

Mario Tokoro: The difficulty will arise when testing a system in its real environment, in order to validate its service in both time and value domains.

Bill O'Riordan: The verification problem is with us for programs running on serial computers: on parallel machines there are a number of complex additional surmountable problems that confront the programmer which will task our intellectual competency and is something we cannot buy from any other vendor.

4.6. Human-computer interface (HCI)

4.6.1. Visualization

Alessandro Giacalone: By visualization one must intend the whole process that takes from sources of data to the delivery of significant information to the individual in visual and comprehensible form. Thus, visualization does not merely mean "user interfaces" as we see them today.

Data and their sources may be of diverse nature. Data may be, for example, numeric, graphical, textual or analogue signals, and may originate from repositories e.g. databases or may be generated dynamically, for example, by simulations or physical processes. Thus, in general, the information contents which are relevant for a certain application must be identified and extracted by correlating potentially heterogeneous data. Finally, decisions must be made about suitable and feasible visual paradigms for representing the information and for enabling efficient action on it by individuals.

Visualization techniques have been utilised already for some time in specific application areas e.g. engineering, entertainment, research, traffic control, however, typically at a high development cost and with ad hoc solutions for each single application. One of the challenges is to develop methodologies that render the development of visualization techniques more systematic and efficient/economical, in order to make visualization available for wider classes of applications.

Nevertheless, research will have to focus on introducing visualization techniques in specific application domains. These may be characterised by the involvement of specific combinations of technologies, type and formats of information, or human interaction paradigms. Examples include the following: visualization of databases, visual information management systems (VIMs), and virtual and computer augmented realities.

Bill O'Riordan: We are second biggest in the world in computer systems for the retail business - and the most important aspect of that business is visualization, because the products will cease to be on shelves - instead one will use all sorts of terminals which will enable a virtual presentation instead. We have four hundred people working on just this project now.

4.6.2. Device-specific issues

Charles Simonyi: There is a great demand for better interfaces, and for new technology such as virtual reality. Perhaps the expectations are too high. At the moment, these types of applications will become available, but the hardware required is too expensive.

Bill O'Riordan: There is also a danger that inadequate technology, especially in the area of virtual reality, will lead to people becoming disorientated by the systems and having accidents because of this. Virtual reality software has a long way to go before becoming as realistic as an aircraft simulator, for example.

Hervé Gallaire: It is difficult to predict some kinds of interfaces because there will be devices for which new types of interfaces have to be invented, such as a personal digital assistant (PDA) and small screens.

Charles Simonyi: We also need to work more on incorporating gestures, such as a sweep of the pen, into input systems. There will also be the head mounted display, and even (I'm not entirely joking) direct input from the brain.

4.6.3. Bi-directional co-operative HCI

Mario Tokoro: User interfaces used to be regarded as for ordering computers to do things and for getting results from computers, but in the future the computer might become something like a friend, so there no need for the user to request something for a communication to exist. This leads to co-operative and bi-directional interfaces, which will be based on gestures, sounds and other means.

Gilles Kahn: Just because we have the ability to gesture doesn't mean we lose our language.

Mario Tokoro: What I term an "intimate computer" can be thought of as an evolved version of a Personal Digital Assistant.

Gilles Kahn: I'm not sure I want my computer to be intimate or affectionate towards me! My cat cares about me because I feed it. This is the type of affection I would like from a machine. I do not want my computer to be like a dog.

Mario Tokoro: A lot of progress has been made on the output side of HCI. However, relatively little has been achieved on the input side. We are working on a system that recognises the expression on your face. For example, when making a database query, the computer will recognise if the output is correct by your expression.

Gilles Kahn: I enjoyed very much the demo I saw of Mario's system, but I have to make some remarks about generalising this idea too much. I have experiments in which people had icons and after a while they replaced all the icons by words. They said that it took two thousand years to get rid of icons (hieroglyphics) and replace them by writing, so why not use words instead?

Charles Simonyi: Icons are not so much used to provide increased understanding, as to save screen space because they are smaller than words. Therefore it is really for expert users that can associate their favourite commands with small icons. But I agree with you that they are not really useful for understandability, although today they are always configurable.

4.6.4. Role of keyboards and voice

Mario Tokoro: Keyboards will continue to be used as an input device, but the usual communication will not be done via keyboard.

Brian Randell: It is naive to assume that voice input will replace keyboards. There will be no demise of keyboards. However, their use will become less pervasive. Even assuming all the predicted improvements in speech recognition, there still will be people working with text via the keyboard.

Tony Temple: The keyboard has been used for two things: getting text into the computer, and as the control mechanism. There are other ways to control the system other than keyboard, such as a mouse and pen. If we are able to dictate at regular speed, straight in to the computer, you may then decide to work in a different way; it would not stop you using the keyboard for certain purposes.

Bill O'Riordan: In some working environments (for example, an open plan office), the using of voice can be a problem.

4.7. New Application Areas

Brian Randell: We are not naive enough to say we can identify new applications, just areas where we think new applications are likely arise.

Gilles Kahn: Multi-media is one of the areas where we do not understand what applications there will be.

Charles Simonyi: The area I wanted to mention was Interactive TV, which I do not think of as multi-media.

Hervé Gallaire: This is pretty general - specific aspects are shopping, library access, on-demand movies, and so on.

Gilles Kahn: What's new about these things is the scale at which it will be. Currently teleshopping I presume exists in the US on a large scale as it does in France with the Minitel system.

Brian Randell: My impression was that there is much more national penetration in France than anywhere else. I think it is much more common for the man in the street in France to do teleshopping.

Charles Simonyi: In the US, teleshopping from Pcs is still very rare.

Brian Randell: Minitel has got embedded into the consciousness in France.

Charles Simonyi: Interactive TV will go way beyond that. For example, sports programs will be enriched by the inclusion of statistics and specific information on the players. When you are watching you would basically create your own commentary and focus on things that interest you.

Charles Simonyi: Another such area is universal messaging, universal communications - you are always in touch. You no longer have an office number and a fax number. Not so much nomadic computing as nomadic life supported by computing.

Charles Simonyi: My theory is that the personal computer card of the future will have just a trivial amount of interface to it and basically all it will be is a memory which contains information which is special to you. As you go around in the world it will imbue artefacts around you with your information and specialise them to you. For example if you put your card near the public phone, the public phone becomes your private phone. It will know your charge number, and all your private phonebook entries and mnemonics.

Tony Temple: Having your personal profile go with you is going to be standard on PCs within the next two years. It will automatically customise any machine you are on, to you.

Brian Randell: Mario, do you have any views from Japan as to what you think new entire application areas demanding, and making demands on, software might be?

Mario Tokoro: One of the areas in which Japan is very strong in games, though games and entertainment will be interleaved. And virtual reality and that kind of thing. There will be a big market for the people who can take a movie, edit it, make a story, make it interactive, and so on.

Gilles Kahn: In the area of publishing there is on-demand publishing and multi-media publishing.

Tony Temple: We have now, in quite a few cities, navigation systems in cars but there is no reason why such a system should be restricted to cars. It can go straight in PCs, and have much wider applicability.

Charles Simonyi: The key thing is that navigation will be an important application. One interesting thing is that a very very complete map of the States is now available in CD-ROM, maps of all the streets, everywhere.

Tony Temple: You can imagine combining this with information - if you want to buy something, it will show you where the appropriate stores actually are.

Charles Simonyi: Or you push a button and the car gives you the cues to go there.

Gilles Kahn: Medicine is another area in which there will be an explosion. This has important implications for software because at the moment we have the software in the instrument, and it is produced by the instrument manufacturer. I cannot believe it will remain that way. The computer will go out of the instrument and will serve to connect a number of instruments, and there will be a market for software - imaging and signal processing. Atlases of the body will be produced - not just standard ones, but ones for individuals.

Tony Temple: Many of the applications we have today are run on a repetitive basis - on a monthly or weekly basis something happens, for example the banks calculating interest. It seems to me that a lot of this will change - and this will involve a huge re-engineering of everything. Similarly with databases - at the moment we run queries to find out what information is in the data base. With triggers and agents, etc., databases are going to alert agents when certain things happen - everything is going to be much more real-time driven. We run spreadsheets at set intervals, but really the spread sheet should be invoked because something has happened, that is, on a real-time basis.

Brian Randell: That fits with Mario's comments about "autonomous agents".

Hervé Gallaire: That's not really a specific application area, but an enabler for lots of application areas.

Tony Temple: Our PCs are generally doing nothing at the moment, just sitting there. Really there should be lots of things going on, sensing things, testing, evaluating, and so on, and making things happen as a result - the majority of which we would never know about. This is no different from what is done with stocks and shares - which are bought and sold without people knowing. But we don't have much of this outside that particular area.

Brian Randell: There are some evident dangers as well as very interesting opportunities in such systems.

4.8. Directions for future research

Directions for future research were implicit in many of the Working group's discussions but a few remarks addressed the issue explicitly:

Charles Simonyi: We need research that will lead to better ability to express domain-specific abstraction mechanisms. Everybody can create abstractions but to create abstraction mechanisms is the privilege of a very few people. Bjorn Stroustrup is almost the only one - and I think that is a bad state of affairs.

Gilles Kahn: Research fuels software's evolutionary cycle.

Brian Randell: We need an increased ability to makes changes to existing programs, and to running systems.

Hervé Gallaire: With the growth of distributed systems it is clear that there is a need for continued research on issues of naming, scaling and mobility.

Brian Randell: There is a need to revisit existing system abstractions to check their continued validity. Developments in hardware and communications are sufficiently predictable that research is feasible on what new software abstractions are appropriate.

The actual summary listing of research topics that the Group drew up was:

- Large Scale Systems - problems of scale, naming, mobility, and so on.
- Components and Glue - abstraction/specialization techniques
- Beyond Objects - autonomy, environmental awareness, emergent properties, intention/desire
- The Incorporation of Legacy Components into Systems

Appendix A
Participants

This appendix contains a brief CV for each of the participants at the workshop.

Laszlo A. (Les) Belady

Director of Mitsubishi Electric Research Laboratory (MERL), Cambridge, Mass., USA

During 23 years with IBM, Mr Belady:

- Led the software engineering effort at the T.J. Watson Research Center (1961-1981)
- Was responsible for software technology at Corporate Headquarters (1981-1982)
- Established the software research program at the Iapan Science Institute (1983-1984)

Prior to joining MERL, Mr Belady was vice-president of the Microelectronics and Computer Technology Corporation, where he founded and headed the Software Technology Program. He is an IEEE Fellow and recipient of the 1990 J.D. Warnier Prize for Excellence in Information. From 1980-1983 he was editor-in-chief of the IEEE Transactions on Software Engineering.

Remi H. Bourgonjon

Director of Information and Software Technology, Philips Research Laboratory, Eindhoven, The Netherlands

Remi Bourgonjon joined Philips Telecommunications in 1972 with an MSc. degree in Mathematics and Computer Science. He started as a development engineer for digital switching systems. He then became involved in the design of programming languages and acted for four years as the chairman of the CCITT working group for this topic. He held various managerial positions in Philips Telecommunications from 1976 onwards, including project manager and department head. In 1984 he became the general manager for switching system developments in the Philips/AT&T joint venture.

In 1987 he became the general manager for the Philips Corporate Centre for Software Technology and in 1990 director of the Corporate Division for CAD/CAM and Software Technology. Since 1991 he has been director of Philips Research Laboratories and responsible for the division of Information and Software Technology.

H. Michael Braude

Senior Vice-President and Chief Research Officer, Gartner Group, USA.

Mr Braude joined Gartner Group as vice president, where he founded the Software Management Strategies (SMS) service. Previously, he served as vice president of technical services at Morgan Stanley & Co., where he was responsible for selection and installation of electronic technology. Before joining Morgan Stanley, Mr Braude was assistant vice president at Metropolitan Life Insurance Co. He also worked for United Aircraft Research Laboratories and taught mathematics at the University of Pennsylvania and the City College of New York.

Mr Braude has a bachelor's degree in mathematics and a master's degree in mathematics, both from the University of Pennsylvania.

Hervé Gallaire

Director, Rank Xerox Research Centre, Grenoble, France

Hervé Gallaire is a graduate of École National Supérieure des Arts et Métiers and holds a PhD in Electrical Engineering and Computer Science from the University of Berkeley, California. He was Professor of Mathematics and Computer Science at SupAéro, head of Computer Science research department at ONERA-CERT in Toulouse. He founded the Computer Science division of CGE corporate research laboratories, as well as the Bull-ICL-Siemens sponsored ECRC research centre in Munich (1984-89). He has been Vice-President in charge of development for Bull (1989-91) for all products apart Operating Systems and Languages, and Vice-President for Software Development with a software house, GSI in Paris (1991-92). His research interests lie in programming languages, databases, modelling and AI, particularly logic.

Alessandro Giacalone

Group Leader (Managing Director from summer 1994), ECRC, Munich, Germany

Alessandro Giacalone holds a Laurea cum laude from the University of Pisa (1977), a Master's (1982) and a PhD (1984) from Brown University. From 1978 to 1980 he was part of the research staff of ITALTEL, Milano. From 1983 to 1984 he collaborated and was Director of Research with IKAN Systems Corp., Providence, Rhode Island, a start-up company concerned with electronic publishing.

He was an Assistant Professor at the State University of New York at Stony Brook from 1984 to 1990, teaching courses in programming languages, compilers, software engineering, programming language semantics and user interfaces. He carried out research in theory and applications of concurrency, programming systems and design/construction of graphical user interfaces.

He joined the European Computer-Industry Research Centre, Munich, Germany, with the position of research group leader. There he started and co-ordinated several research projects concerned with concurrent and distributed computing, multiple-paradigm programming, graphics and visualization, teleconferencing. He also contributed to the development of activities concerned with the analysis of perspective technological and market scenarios. He will become managing director of the centre in the summer of 1994.

Brian Gladman

Director of Communications and Information Systems Engineering, Ministry of Defence, UK

Brian Gladman gained a degree in Theoretical Physics from Imperial College in 1964 and later a PhD for work in microwave antenna systems. His research work in phased array antennas continued from 1964 to 1972 at the Admiralty Surface Weapons establishment with a three year period on exchange at the US Navy laboratories in San Diego. In 1972 he moved to lead research on computer based command systems and was appointed to Head of Division in 1976. In 1981 he moved to lead defence computing research at the Royal Signals and Radar Establishment, Malvern. In 1988 he moved to become the Director of Strategic Electronic Communications within MOD and became Director of Communications and Information Systems Engineering after a reorganization.

During his career with MOD Brian Gladman has played a leading role in defence computing where his particular interests have been in software engineering and high integrity systems. He has a particular interest in information security in globally distributed information systems.

Andrew Herbert

Chief Architect, ANSA, Cambridge, UK

After graduating from Leeds University, Yorkshire, in 1975 Andrew Herbert studied at Cambridge University and gained a PhD for work on hardware support for capability-based protection systems. He joined the Cambridge faculty and took a leading role in the development of the Cambridge Distributed System, an early example of a processor farm and supporting network servers.

In 1985 he left Cambridge and became the Chief Architect of ANSA, an international consortium of computer vendors, telecommunications operators and end users developing architecture and technology for open distributed processing. Particular successes of ANSA have been the transfer of architectural concepts into standards, advanced prototypes of subsequent industrial software technologies and a strong contribution to the software strategies of the ANSA sponsors.

Dr. Herbert is a liveryman of the Worshipful Company of Information Technologists, a Chartered Engineer and a member of the BCS and IEEE.

Cliff Jones

Professor of Computer Science, University of Manchester, UK

Cliff Jones was appointed Professor of Computing Science at Manchester University in 1981. In 1988 he was awarded a five-year SERC "Senior Fellowship" which permitted him to concentrate on research into concurrency. His computing career began in the 1960's in industry; during a period which included two spells in the IBM Vienna Laboratory, he made a major contribution to the creation and dissemination of the formal development method known as "VDM".

Since moving to Manchester he has followed two main lines of research:

- Support systems for formal methods (including the 'Mural' theorem proving system).
- Concurrency.

His early research on concurrency showed how the notion of interference could be handled in specifications and proof obligations, but the number of proofs in arbitrary shared-variable programs made the approach difficult to use in practice. Professor Jones has written a series of papers which show how concepts from object-oriented languages can be used to constrain interference in a way which makes a compositional development method practical.

Gilles Kahn

Scientific Director, INRIA, France.

After graduating from the Ecole Polytechnique in Paris in 1966, G. Kahn worked for two years at the French Atomic Energy Commission computer center before going to Stanford University. Returning from Stanford at the end of 1971, he established a research group at INRIA with J.M. Cadiou and worked on models of parallel computation. In 1974, with G. Huet and B.Lang, he began work on programming environments. Since then, his interest has been shared between:

- Theoretical computer science foundations of the semantics of programming languages.
- The development of practical systems (such as Mentor and then Centaur) that embody the best theoretical ideas.

Currently, he is particularly interested in computer proof assistants and the technology needed for concurrent engineering.

In 1983, G. Kahn moved to the south of France to help in establishing the Sophia-Antipolis laboratory of INRIA. Since December 1993, he has been a Scientific Director of INRIA. G. Kahn has taken an active part in the creation of several of INRIA's subsidiaries. In particular, he sits on the Board of Directors of ILOG and Connexite. INRIA has been at the origin of about 20 start-up companies, most of them in the software industry.

G. Kahn is a member of Academia Europaea and the 1992 recipient of the Monpetit prize, given by the French Academy of Sciences.

Mike Lesk

Michael Lesk obtained his BA in Physics and Chemistry in 1960 and his PhD in Chemical Physics in 1964, both from Harvard University.

He is now manager of the Computer Science Research Division at Bellcore. For the previous fourteen years he had been a member of staff in the Computing Science Research Center at Bell Laboratories. From 1983-1985 he was also adjunct lecturer in Computer Science at Columbia University. During most of 1987 he was visiting University College London as a Senior Research Fellow of the British Library. He is Visiting Professor of Computer Science at UCL and also Visiting Fellow in the school of Library and Archive Studies at UCL.

His current research involves the creation of an on-line chemistry library. Recent interests include access to data bases, computerized route finding, and the comparison of menu-oriented and command-oriented human interfaces. Other projects include computer typesetting tools, electronic mail, computer networking, and compiler generators. His areas of research have included artificial intelligence, information retrieval, compilers, human factors, networking, and data bases. Among the software tools he has written are the original versions of the UNIX utilities **uucp**, **tbl**, and **lex**.

Jessica Litman

Professor of Law, Wayne State University, USA

Professor Litman obtained a BA from Reed College and a M.F.A. from Southern Methodist University, before obtaining her Juris Doctor degree from the Columbia Law School in 1983. She was Law Clerk to the Hon. Betty B. Fletcher, of the United States Court of Appeals for the Ninth Circuit, Seattle, Washington from 1983-84. She then held academic positions at the University of Michigan Law School: as an Assistant Professor from 1984-87, and as an Associate Professor from 1987-90. She then moved to the Wayne State University Law School, where she became Professor of Law in 1991.

Professor Litman specializes mainly in the subject of Intellectual Property Rights and Copyright, one of her areas of particular interest being software. She has been a Member of the Board of Trustees of the Copyright Society of the USA, Chair of the AALS Section on Intellectual Property, and Member of several committees of the American Intellectual Property Law Association.

Bill O'Riordan

Head of Research, ICL, UK

Professor O'Riordan is a visiting professor at Imperial College of Science and Technology, London. He is also an active member of the British Computer Society and a builder/designer of helicopters, in addition to his role as Head of Research at ICL.

Chris Phoenix

Vice-President of Technology Strategy for ICL Client-Server Systems, ICL, UK

Chris Phoenix has degrees in Physics (BSc Hons. Manchester) and Operations Research (MA. Lancaster).

Chris has over twenty years experience in the IT industry. In that time he has undertaken activities varying from hardware and software development through to technical and project management roles on a variety of customer application developments. These have included Health Service and Defence projects and assignments in the UK and overseas. From 1985 to 1989 he was a Director of STC's Corporate Research Facility. In 1989 he transferred to ICL. He is now Vice-President of Technology Strategy for ICL Client-Server Systems. This organization employs over 2000 staff and accounts for over one billion of ICL's turnover. In this role Chris is responsible for overseeing the technology acquisition strategy for all of ICL's PC and open systems platforms and for its generic client-server applications software business.

Chris sits on the Shareholder's Council of the European Computer- Industry Research Centre (ECRC) in Munich and is Chairman of the Management Committee of the ANSA Project in Cambridge which is developing the ANSA architecture for distributed computing. He is also the ICL representative on the Imperial College Computing Forum Advisory Board.

Brian Randell

Professor of Computing Science, University of Newcastle upon Tyne, UK

After graduating in Mathematics from Imperial College, London in 1957 Brian Randell joined the Atomic Power Division, English Electric Company Ltd., where by 1964 he was head of the Automatic Programming Section. In 1964 he joined IBM and worked mainly at the Thomas J. Watson Research Center on the design of an ultra-high-speed computer, and then on an investigation of the design of both the hardware and software of a large multi-processing system.

In 1969 he took up his present position as Professor, and Director of Research, at the Department of Computing Science, University of Newcastle upon Tyne. At Newcastle he has been Principal Investigator on a succession of research projects in reliability and security funded by the SERC, MoD and Alvey Programme, and is now Project Director of the ESPRIT Basic Research Project on Predictably Dependable Computing Systems. Professor Randell is a Fellow of the British Computer Society, and a Chartered Engineer.

Gill Ringland

Finance & Business Strategy, ICL, UK

Gill Ringland graduated as a physicist but, after two years at the University of California at Berkeley wrestling with early versions of OS/360, in 1969 joined the then-young software house CAP, becoming Chief Technical Consultant. After work on the Computing Science Committee of the UK Science and Engineering Research Council, she was a member of the US/European semiconductor venture Inmos on startup. She then became Software Director for Modcomp, before joining ICL where she has had responsibility for database, network management and office systems products, internal IT, and more recently, business strategy. In connection with this latter responsibility, she initiated this Workshop as a result of work to establish a long term planning framework for ICL, having found that, while there is much speculation, discussion and consensus surrounding the changes in hardware capability and price, and the effect of this, there is much less about software.

She is a member of the Worshipful Company of Information Technologists, in addition to the BCS and IEEE.

Charles Simonyi

Microsoft, USA

Charles Simonyi was born in 1948 in Budapest, Hungary. During 1966-67 he was at A/S Regnecentralen, Copenhagen, Denmark, where he worked on the RC 4000 operating system in a team of three led by Per Brinch Hansen. After receiving his BS in Engineering Mathematics at UC Berkeley in 1972, he participated in various computer enterprises including Berkeley TSS, Berkeley Computer Corporation, ILLIAC-4.

He was at Xerox PARC in 1972-1980, where he developed Bravo for Alto, the first WYSIWYG word processor. He received a PhD in Computer Science at Stanford in 1976, for a thesis entitled: "Meta-programming: a Software Production Method".

Charles Simonyi joined Microsoft in 1981, to start the development of microcomputer application programs. He hired and led teams developing Multiplan, Microsoft Word, Excel, and other applications. This part of Microsoft's business has grown since to about two billion dollars a year. In 1991 he joined Microsoft Research, where his main interest is Intentional Programming: a user-extendible programming language paradigm which strives for maximal reuse of components by separating abstract computational intent from implementation detail.

David Talbot

DGIII, Information Processing Systems and Software, ESPRIT

David Talbot graduated in Mathematics from Oxford in 1960 and joined ICL as a technical trainee. He worked on a variety of early computer systems and with many of ICL's first users. He held management appointments covering technical, business and marketing activities and for four years worked on secondment to Department of Trade and Industry, as the first Director of Software Engineering in the UK Alvey programme. To occupy spare time he had the pleasure of acting as a non-executive Director to a Cambridge start up specializing in the design and production of top of the range DSPs. He has been with the European Commission since 1989 where he is responsible for Esprit's activities in software and advanced information processing.

He is a liveryman of the Worshipful Company of Information Technologists.

Tony Temple

IBM Warwick, UK

Early days began at IBM's Banking Branch in the city of London, developing numerous commercial systems in partnership with leading financial establishments. Involvement in software development increased when transferring to IBM's service organisation to lead the creation of a package for analysing market research data. This product became a market leader in many countries.

In the latter 1960s, IBM launched a Time Sharing service. Tony was involved in the initial technical decisions and was a founder member of a new European organisation. Having initially specialised in Analytical applications and in particular software development, he became manager of the development function, followed by overall manager of the UK Time Sharing service and development organisation. This experience provided expertise in End User Computing and development skills in associated software.

During the early 1970s, Tony established a team to develop a set of integrated software tools for Management information and Decision Support Systems. The residual offering named Application System, became a market leader throughout the world and is today one of IBM's premier software products. This success led to the establishment of an International

In the mid-1980s, Tony took an assignment in the USA to lead the design of IBM's future office systems. As part of this project, he led the definition of IBM's User Interface Standards. Much of this design and architecture has subsequently been adopted by the computer industry and is reflected in the implementation of most popular Graphical User Interfaces.

Current business responsibilities span User tools that focus on transforming data into information. These tools include Query, Forms, Decision Support and Management Information Systems. This work is performed at a number of IBM laboratories in Europe, USA and Japan.

Two of IBM's European Laboratories, Warwick in the UK and Dublin in Ireland, directly support this business area and formally report to Mr Temple.

Technical responsibilities are somewhat broader covering the strategy and technical definitions of IBM's Workgroup Architecture. This relates to both client and server functions with particular emphasis on the user interface and overall integration of function. This work is supported by his IBM fellowship, where is researching and defining much of IBM's user Interface Architecture for the latter part of this decade.

Mario Tokoro

Keio University, and SONY, Japan

Mario Tokoro is Professor of Computer Science in the Faculty of Science and Technology, Keio University, and is currently the head of the department. He is also the director of Sony Computer Science Laboratory, Inc., since its foundation in 1988. He has been in various research areas including Computer Architecture, Computer Networks, Object-Oriented Programming Languages and Operating Systems, and Artificial Intelligence. His largest interest lies in the development of computational models/paradigms for open, distributed, asynchronous environments. As such a model, he proposed the Computational Field Model, which regards networks of computers as a continuous computational field, where real-time software agents inhabit and occasionally co-operate.

Peter Wharton

Chief Engineer & ICL Fellow, ICL, UK

[Peter Wharton intended to come to the workshop but was taken ill at the last minute. He submitted a position paper and this is included in Appendix B.]

Peter Wharton gained a mathematics degree at Leicester University and went straight into the computing industry, joining LEO Computers in 1962. Following LEO's take-over by English Electric, he was seconded to RCA in the USA for two years. At RCA he initially worked on the evaluation of the emerging IBM 360 Architecture and later on the design of a virtual store system. On returning from his secondment, he joined the System 4 development team and was responsible for the design and implementation of System 4 Indexed Sequential. Several mergers later, he found himself working on the New Range product in Putney and this was followed by a move into the VME operating system central design team (OSTECH). He became manager of OSTECH and held this post for several years where he had full responsibility for the strategy and design of VME as well as the definition of the software development system (CADES). Subsequently, he set up and managed Systems Architecture, the central design team responsible for the development and introduction of Series 39.

He is a member of the Salford IT Research Advisory Committee and the DTI/SERC JFIT Standard, Security and Quality Committee. Currently, he is Chief Engineer in Corporate Systems Division working in a corporate role, covering both systems engineering methods and processes and the identification and evaluation of future technologies.

Robert Worden

Chairman, Logica Cambridge, UK

Dr Worden obtained his MA and PhD from Cambridge University. After doing research in particle physics at CalTech, Rutherford lab and CERN, Robert Worden joined Logica in 1975, managing a project developing an interactive simulation of electronic warfare. During this project he designed and implemented a relational database management system, which after further development was sold internationally by Logica as the product Rapport. He managed this activity to the point where the product won an ICP $5M award.

In 1984 he moved to Cambridge to help set up Logica's R&D facility, Logica Cambridge. He managed a 20-person ESPRIT project to develop an expert system in electromyography, and was then managing director of Logica Cambridge, and also does technical review of Logica's major projects world-wide. He served on the Alvey IKBS Advisory Board, and on a number of MoD, SERC, and DTI advisory groups.

William (Bill) Wulf

AT&T Professor of Engineering and Applied Science, University of Virginia, USA

Before taking his present position at the University of Virginia, Dr. Wulf was Assistant Director of the National Science Foundation (NSF) where he headed the Directorate for Computer and Information Science and Engineering (CISE); NSF is the US Federal agency responsible for funding academic basic research.

In 1981 Dr. Wulf founded Tartan Laboratories and served as its Chairman and Chief Executive Officer; prior to that, he was Professor of Computer Science at Carnegie-Mellon University. Professor Wulf's research interests span operating systems, optimizing compilers and computer architecture. Dr. Wulf is a member of the US National Academy of Engineering, a member of the Council of the ACM, a Fellow of the IEEE and AAAS, and Chairs the Computer Science and Telecommunications Board of the National Research Council (the operational arm of the US National Academy Complex).

Appendix B

Position papers

This appendix contains position papers from delegates. Drafts of nearly all of them were distributed before the workshop and, to some extent, formed the basis for discussion. Some material from these papers has been included in the main body of the report.

Les Belady

Background

In the late sixties most computer applications were of the scientific/engineering or accounting type and presented themselves mathematically expressible, so that programming was essentially the translation of formulae to computer processes, However, it was observed that the required relatively simple way of programming did not work well with software which was too large or complex, or when the problem was not already formalized. The set of programs needed for the computerization of a hospital would be a good example. To produce this software takes more than the translation of a mathematical formula to a machine interpretable code. This action must be preceded by design, an engineering activity to formalize the often fuzzy problem requirements such that programming proper can start.

As a result, in 1968 the now famous NATO conference in Garmischpartenkirchen launched a struggle to convert programming into an engineering discipline. The goal was to find a set of teachable rules which then guide the software engineer's efforts through the maze of schedule, budgetary, safety and other constraints.

There was another reason for a more disciplined approach. By the time of the conference some software projects themselves became so large and complex that a single, though quite capable, person could not do the job anymore within reasonable time. Programmers had to team up, but sometimes even the team was not enough and we witnessed the appearance of mammoth projects of hundreds or even thousands of "programmers".

This inevitably led to software specialists organized around a "process", consisting of consecutive phases such as requirements, design, implementation, testing, and so forth. And while most large programs live "Long after they are first put to use, they also keep being modified, which in turn makes them less Modifiable. As a result, maintenance becomes a nightmare and often the most significant software cost component.

To co-ordinate the many interdependent activities of software development and maintenance, management came to the forefront as the primary means of improving the productivity of the process and the quality of the resulting software product. Even today, for many people software engineering is essentially a management discipline. (See Boehm's book on Software Engineering Economics.)

Evolution into systems integration

Today even this model of software engineering is no longer adequate; it will no longer work. Why? Because of the nature of recently appearing new application systems with an added dimension of complexity: communications and networking. This leads to new kinds of programs:

We have seen that this world is undergoing a shift in mission that will utterly transform the software engineering discipline. Because integration is the key to creating large, complex computer systems, software engineers must become application system designers, and ones with significant hardware and application domain expertise at that. These designers must take, an often staggering number of components, many of which are themselves quite large, and combine and recombine them into The integrated networked super-applications of tomorrow.

- They are *distributed across networks*. Hundreds-sometimes thousands-of computers are networked together so that their users can work on common problems. Typically, hundreds of individual processes are executing concurrently and interacting with one another to produce the desired results.

- The systems are *domain specific* - indeed, often enterprise-specific-meaning that they have not been bought off the shelf but tailored, through the collaboration of their designers and their users, to the needs of those working in the particular endeavour they have been designed to support.
- Many different *applications* have been, integrated to meet a variety of needs - needs for word processing, for spreadsheets, for flight simulation, for accounting, whatever - into one comprehensive system, each part of which is accessible to all users.
- Finally, a set of *heterogeneous hardware platforms* and workstations have been networked together in an integrated way. In other words, users of Macintoshes, for example, can share data and communicate with those having PCs, and the system functions smoothly.

The key here is *integration-of* applications, of hardware components, and ultimately of the people who use the system to work together across a network. And fundamentally, it is software that is the agent holding together all these disparate elements: as I have claimed in other contexts, software is the glue of large systems Creating this software requires dealing with immense complexity both of the systems themselves and of the process of designing them - a perspective illustrated in simplified form in Figure B.1.

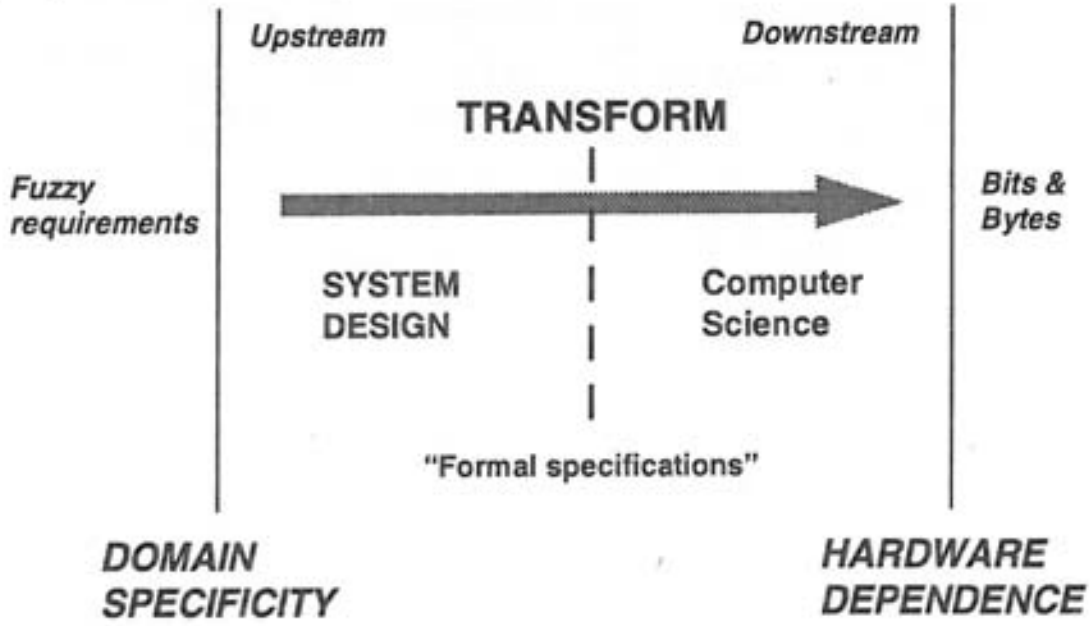

Figure B.1. From software engineering to system design.

Figure B.1 shows that there are elements of the process that did not exist prior to the advent to today's large systems: the need for domain knowledge in the early, upstream phases of design, and the need for hardware knowledge in order to integrate the system across platforms. These new elements are the points where software engineering and knowledge engineering intersect: the software engineer becomes an *application system designer*, an engineer not just of software but of hardware as well, who must understand the application domain in order to select the best software and hardware for a design's implementation. Perhaps most importantly, the engineer must be able to work well in teams with others, because a task of this complexity requires the expertise and co-operation of a wide array of specialists-specialists in the application domain, in real time performance, in reliability, and in many other areas.

Quite obviously, there is now a level of complexity in the software design process that has not heretofore existed. The complexity encompasses both people who must now work together in large, often, distributed and usually interdisciplinary project teams - and the software itself. It is this latter complexity that of software - that I turn, now to address.

Because these integrated, networked application systems must both offer a wide variety of services and be tailored to the particular needs of those using them, we see a split in the domain of software. I talk about this split in terms of Type A and Type B software. Type A is software

that is, essentially, purely some component of the new system: an individual application, an off-the-shelf subroutine, a compiler, whatever. Typically, this type of software can be created anywhere, and is not particularly dependent on the specific domain in which the system will be used.

Type B software, on the other hand is the glue, the system's traffic controller, the software that holds the components together and integrates them into a smoothly functioning, domain-specific system. For instance, large Type A applications-computer-aided design, computer-aided manufacturing, inventory control, and so on-may be integrated by Type B software into a large enter prise-wide network. Usually created jointly by the software engineer and the customer, Type B software requires an understanding of the application area that only domain specialists can provide. Unlike its Type A counterpart, it cannot be created "offshore": in effect, this is the software that constitutes the fundamental nature of the system itself.

Type B software is where our biggest challenge lies. Its design requires a level of interdisciplinary collaboration and sheer technical mastery that is unparalleled in our industry. But with this challenge comes opportunity in equal measure. In return for mastering the complexity of Type B software, competing nations and enterprises could achieve a tremendous advantage in the global market for large, complex, integrated systems. And as we accumulate the know-how required to create this software, our growing expertise will constitute an investment by industry that will be hard to match elsewhere. Figure B.2 sums up what I have been saying, showing how the world of software has changed and will continue to change. We have moved from the process of developing software, which relied heavily on applications and systems programming, and toward the systems design process, which demands the creation of Type B software to integrate the components of a large complex application system. Again, we need knowledge engineering to help us design this new software, because software engineering as it has been practised will not suffice.

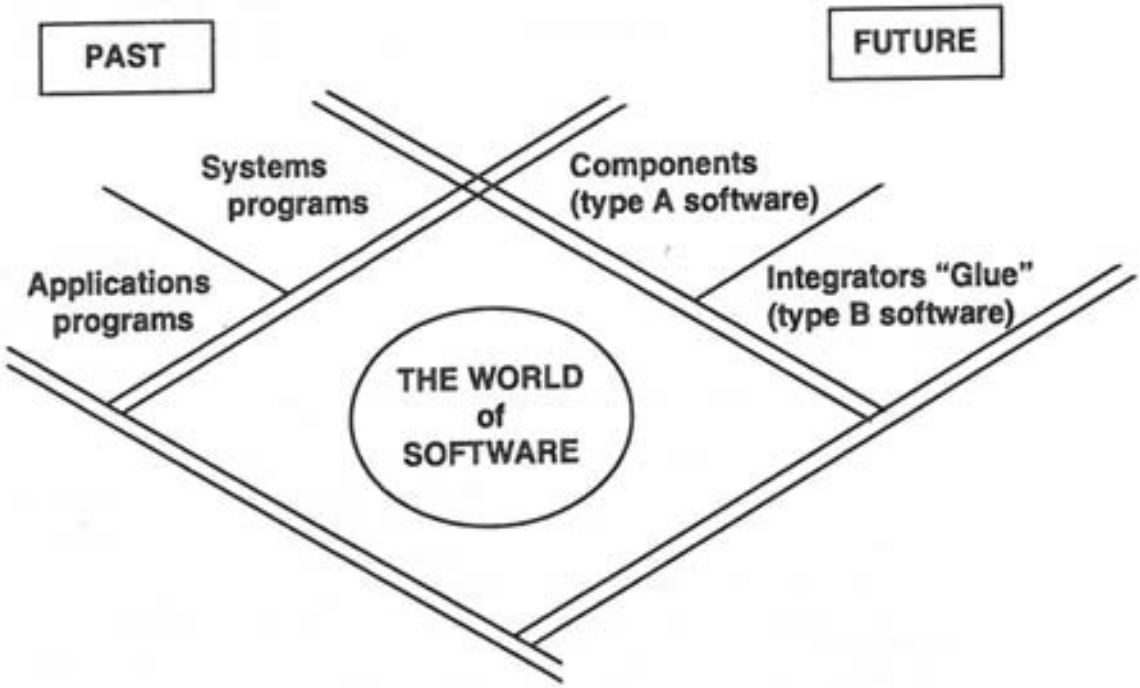

Figure B.2. The world of software: past and future.

What avenues exist for supporting this system integration know-how? Answering that question fully would lead to a paper in itself, but for our current purposes here are some suggestions that spring to mind:

- We need increased standardization of - Type A software to optimise the design of Type B software and to make it more economical. A recent example of this trend is X Window Systems, whose representation can simply be assumed by a new system's designers, The designers are then freed to work on the system's particular, custom-tailored modules or features. Ultimately, a standardised interface for all stand-alone applications also lowers

the cost of creating new systems: the more people who are using a given component such as X Windows, the lower its cost will be.

- Extended software engineering *education and retraining* are required as well. As I mentioned earlier, no longer is it sufficient for the software engineer to simply have a degree in Computer Science; to design tomorrow's large systems, he or she must have more than a passing familiarity with hardware and communications technology, know how to design complex hardware/software configurations, and have some expertise in at least one application domain, such as manufacturing or banking.

- Finally, again, the ultimate goal of knowledge engineering is to computer aid the system design process - not just the software design process, but the process of designing these large, complex, networked application systems.

This latter point brings me to the final section of this paper, a brief discussion of two of the research areas which are currently exploring ways to computer aid the process of designing systems: computer supported Co-operative work (CSCW), and Visualization.

CSCW sprang from the awareness that while the first decades of computer use supported only the individual - with the machine used interpersonally only for purposes of communication, at best developments have evolved toward assisting people as they collaborate and work together. This new research area held its first, very interdisciplinary conference in Austin in late 1986; this conference was only the first of what has become a string of events where people discuss their work in CSCW. One recent thrust is groupware, which supports on-line collaboration via networked computers or workstations. Examples include the co-authorship of complex documents (such as system specifications) and such efforts as COLAB at Xerox Parc. Using groupware, teams can use the computer to collaborate across distances or, as in the case of shared work surfaces for conferencing, in the same room.

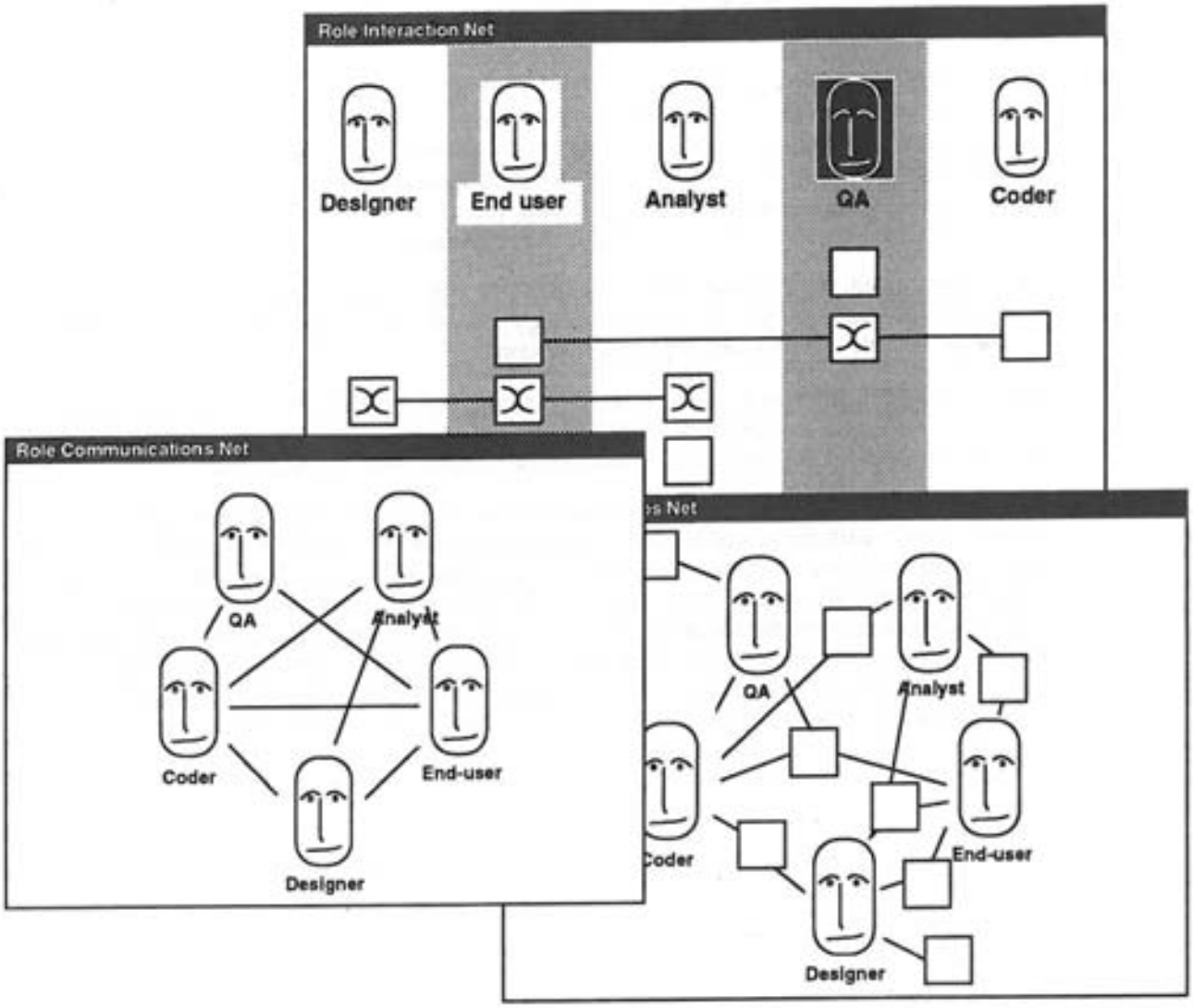

Figure B.3. Users of MCC's Co-ordination Technology can capture and edit complex processes.

Groupware tools typically support direct, session-oriented collaborations that occur at discrete points in time. Another important area of CSCW research, Co-ordination, is somewhat different in that it supports large, complex projects over longer periods - often over months, or even years. Essentially, Co-ordination systems seek to computer assist multifaceted human activities, such as planning, designing, and implementing complex processes - ones which involve, for example, both people and machines. Many of these systems, such as the Co-ordination Technology system under development at MCC's Software Technology Program (see Figure B.3), provide tools and methodologies for capturing existing processes and designing new ones, as well as for viewing and editing these evolving processes.

How might computer aided co-ordination of people and resources enhance the process of large system design? For one, studies have shown that the quality and productivity of the group depends more on interaction among people than on the sum of all individual actions. Co-ordination systems consider a project (or some other organized human activity) to be a distributed, asynchronous amalgamation of actors, roles, teams, actions, and interactions. The systems then order these elements so that people know when events are to occur, when an interaction between roles is meant to accomplish, and how interactions affect other interactions. Because scheduling and development are integrated at the workstation level, a project's management load is lessened, leaving more time for motivation and creativity. There is a better exploitation of concurrency, leading to a reduced "product" cycle, as well as better documentation of the process involved in a complex project. This latter advantage allows for

greater transferability of accumulated experience, as well as better training of new staff via computer models of the project.

Systems Visualization is another knowledge engineering research area that holds great promise for the world of large system design, in that it increases insight into complex systems. Already, Visualization is a major feature of many scientific and engineering applications: scientific researchers have found that "what if" computer simulations, ones that allow the user to visualize hypothetical results, are often far more useful than concrete experimental equipment. An example is, of course, wind tunnel visualization.

In the context of software system building, we have a somewhat thornier problem, however: the problem of visualizing the non-tangible. What, for example, is the shape of software? Traditional flowcharts are not particularly helpful for quickly recognizing interesting characteristics. Perhaps standards for software shapes, like standards for windows, will help. Another problem is how to visually represent the dynamics of a system. Currently, the use of displays is little more useful than the traditional "sheet of paper" mode of System Visualization, and animation and artificial reality programs, however tantalizing, are still slow and cumbersome. These examples show, however, that a good amount of excellent research is underway in the area of Visualization, and I predict that results will soon have a tremendous impact in the world of large system design.

We have seen that this world is undergoing a shift in mission that will utterly transform the software engineering discipline. Because integration is the key to creating large, complex computer systems, software engineers must become application system designers, and ones with significant hardware and application domain expertise at that. These designers must take an often staggering number of components, many of which are themselves quite large, and combine and recombine them into the integrated networked super-applications of tomorrow.

Summary

In the long run, the engineering of software of Type B will not be a speciality by itself. It will be the sub-discipline of many professions: medical, legal, management, engineering. It will be the skill needed to design computer aided systems in virtually every human endeavour.

Remi Bourgonjon

Basic assumptions

In the coming years, the price/performance ratio for micro-electronic products will decrease with at least the same speed as now. The consequences are that many more, especially consumer products, will have enormously increased functionality implemented by embedded software (already now there is 0.5 Mbyte software in an expensive TV and 2 K software in an electric shaver).

Technology issues

As a result of my assumptions, important technology issues arise:

- mastering complexity; many products will become part of systems and need well thought of (open?) system architectures. Keeping complexity under control will be a major issue;
- application oriented programming paradigms; current software engineering technology may not be adequate to achieve the required productivity and reliability;
- user interface technology; increased functionality of products will need much more considerations for adequate user interfaces.

Market and customer issues

Due to the very much increased potential functionality of products, it will be much more difficult to forecast market successes and customer acceptance of new products. Therefore, flexibility and innovation speed based upon a stable set of core concepts will be essential.

Instability in business strategies

Software technology will allow many new players to attack existing businesses in a completely different way ("information highway", electronic shopping etc.). It will upset many current business strategies.

Vulnerability of society

As society will become more dependent on software, there is the big risk that nobody will oversee the total system and dependencies between the different components. This will leave much room for malicious people to harm society. Unreliable software will have the same effect.

Mike Braude

This paper was written just after the Workshop and is included as a position paper as well as being circulated to Gartner Group subscribers.

Software 2000 - Another View of the Future

Building up the research network and accessing disparate views are crucial parts of the research process. Accordingly we were happy to join "Software 2000", a workshop held April 6-8 near our Windsor, UK office and sponsored by ICL and Esprit. This meeting aimed to follow-up a "1968 NATO" conference, at which the term "Software Engineering" was coined in a sarcastic sense; the participants were amused when it caught on positively. The main result of that meeting was recognition of a software crisis, the inability to design and build large software systems.

It is my sense that crisis still exists, differently because the times are different (in scale, sunk investments, changed IT mission, computing models). It is not perceived as a cause for alarm because everybody has adjusted to it, much as our economy has adjusted to a large federal budget deficit. It is a bit frightening to ponder the thought that (the structure of) our industry depends much on its inability to write high-quality software quickly, just as our economy needs the deficit. Anyway, a sense of crisis did not dominate "Software 2000".

"Software 2000" had 22 attendees, including five Professors, eight heads of research laboratories, one senior staff member of a software company (Microsoft) and one Chairman of a consulting company. Seven nationalities, including Japan were involved. The group, and the discussions, were high-powered but warm.

The overall message (my interpretation) of the event is that the twin thrusts of software/programming ubiquity and a thick information infrastructure (unified networking of information, entertainment and embedded processors) will change the civilized world, in ways we can but begin to comprehend. Telecommuting, effective 1-6 person Companies and the leaps into existence of interest groups on the Internet are just initial and (comparatively) trivial results. What will happen after the computers in your cars, home appliances, televisions and telephones are networked and programmable by you? The current heat in the wireless and mobile computing arenas is only a prelude to that drama. Considering that the desktop drive caused a drastic dislocation within corporate America (and Europe), how the impact of these next two visible shifts be handled?

Even now the number of "intermittent" programmers, at (we estimate 20 million world-wide) dwarfs the total some 3 million (CG estimate) software professionals. This amazing statistic already has severe implications, one of them quality - a study known to one of us showed that 80 percent of non-trivial spreadsheet applications have significant bugs. Therefore almost all of you are using radar screens with false images. While the number of software professionals will increase to 13 million by 2010, the "amateurs" group will explode to 200 million. What on Earth will they be programming? What will the platforms be?

The "Software 2000" group projects a large number of domain specific software frameworks; communications; entertainment creation (authoring); visualization; information retrieval. Frameworks consist of applications programming interfaces, languages, tools, etc., and I won't insult these frameworks even before they are born by bringing in the term "architecture". These frameworks will evolve and adapt very fast, and will each give rise to a software industry. Will one or more of these framework be monopolistic? Will any one nation be positioned to challenge that monopoly or cartel? How will intellectual property rights be defined and protected within these frameworks? Will these frameworks be able to escape, as have books, strict liability consequences? How will pornography and very harmful criminal activities be addressed?

The above paragraphs summarize just one part of the "Software 2000" proceeding. Proceedings will be published in several months, and I will advise you how to obtain them.

Hervé Gallaire

Notes

The first issue with SE is a meta issue: there is no understanding of the real drivers which control any improvement to the practice of SE. We know there are many such drivers, among which the following list is all but complete: skills, tools, environments, organization of work and work practices, leadership, process management, methodologies, participatory design and user involvement, role of standards, role of technology, etc. Lacking this understanding, SE solutions are incomplete, at best. We do not know whether we work on the appropriate problems. This creates a space where it is hard to repeat best practice, which is unfortunate, because best practice is, with technology and modelling, at the core of what happens in other, more successful, areas of engineering.

There is a mismatch between the modelling concepts offered by our tools and the models of the real worlds we work with and in which software is going to play. This mismatch has three components: representation, computation and composability. This mismatch must be reduced.

We do not understand reuse well enough; we expect too often total reuse from a reuse approach, independently of the context in which it appears.

Design methodologies are poor, in general: OO design, Client/Server design, distributed or parallel system design, User Interface design, real-time design are examples of domains in which we have very little at hand.

We do not understand well the analytical aspects of SE, even when it's possible to develop them; e.g. we do not really understand the performance issues of various database techniques, or of various networking schemes, even though this should be possible.

Lack of real standards, even in well understood areas such as SQL and C++

Position paper

Abstractions, modelling, and composability: three building concepts for SE in the real world

Software Engineering rests upon a few stable concepts, elaborated over the years. In the formulation phase of the concepts that led to SE , abstraction and modelling were to be used in tandem. This has not worked out as expected, particularly the modelling approach to SE has had limited impact. In this short note, we will review the role of abstractions and of modelling. It is clear that the practice of SE gives a prominent role to abstractions, much less so to modelling. To date, our success has been limited. We have not built reusable high-level abstractions, the abstractions have not been very composable, and we have rarely reached a level at which very much of our systems seem like a natural model of the real world behaviour we are after. Our perspective is that by focusing on basic issues in abstraction, modelling and composition, we can radically improve our approach and practice of SE, in general as well as in specific domains of applications. We argue that a promising strategy consists in applying powerful compositional tools to simple, effective, i.e. computable, models of small chunks of the world. We'll briefly describe three pieces of work we are doing in this area after analysing the shortcomings of the current solutions in each of these key domains. All our approaches provide us with new insights in the long lived debate about declarative vs. procedural programming.

Abstraction is a very powerful concept; it can be asserted that it has been the major contributing factor in the evolution of SE; object-oriented technology, the latest addition to the SE framework, has been promoted largely because it supports abstractions to a greater extent than other programming technologies. By their very nature, abstractions allow us to adopt a view of a system that brings salient issues into focus while hiding issues of less importance. We have used this to achieve "information hiding" or "abstraction boundaries". The abstraction and its modular implementation behave as black boxes, irrevocably hiding certain choices behind the abstraction boundary; this is indeed what users want, provided that the abstraction reveals,

above the boundary and through its behaviour, what they are interested in, exactly. If it does not, and it rarely does, users of the abstraction, i.e. programmers or designers, have either the choice of programming around it, leading to obscure and redundant code, or have to redesign and redevelop new abstractions, defeating the major purpose of using abstractions, i.e. to enable reuse of design and of code.

Whoever has been involved in the design of large pieces of code, or of libraries, in the most modern object-oriented language, will undoubtedly have experienced the difficulty of designing for reuse; constant trade-offs have to be made to achieve this goal but these trade-offs are not apparent anymore to the users of the objects to be reused. We believe that there is now evidence, from the work conducted here and in other places, that it is possible to design "multi-ported abstractions" that honour the concept of abstraction, and yet do this in ways that makes it possible to later go back and, through a parallel abstraction, address some of the previously hidden issues. So, we can define a very high-level modelling language that normally hides its implementation issues, and still be able to go back and access those implementation issues when need be. We can do this, for the time being, in limited contexts, yet with very wide applicability. The discussion will sketch the work conducted at PARC on Meta Object Protocols, particularly in the area of programming languages. The idea here is that the programmer bases his or her approach on separation of concerns: concern about the application and concern about the behaviour of the underlying tool; this is obtained through the interplay of three technologies: reflection, object-oriented programming and incremental specialization. It is worth noting that the industry is starting to adopt such approaches, e.g. to the problem of memory management, although these techniques do not have the sophistication, nor the power, of what will be described.

Modelling, as a way of developing software, has not been successfully incorporated to SE technologies, with the exception, to a limited extent, of SE in database design. Modelling presupposes a level of abstraction, but for our purpose, it differs from abstractions, in several ways; it requires at least that some reasoning capabilities about the objects being modelled be available. It derives its value from capturing, in a multi-usable way, essential regularities (facts, assertions), about the domain the computational activity is about. The root of the comparatively low use of modelling techniques in today's practice of SE, as compared to the use of abstractions, is clearly due to the fact that we haven't had coherent, integrated methodologies for developing complex computational systems based on modelling ideas. Generally applicable reasoning techniques have been emerging only slowly, and mostly in well understood areas of mathematics, such as type theory or linear programming. This is not a criticism of a whole field of theoretical computer science, which aims in part to provide such tools, but it is the reality of SE practice, even if in areas such as testing, the use of reasoning capabilities becomes slowly a reality for safety critical application domains. The tools we have had, such as programming languages, have embodied little, if any, significant modelling power; the capabilities made available speak little in terms of the real world the user is interested in, although the user is encouraged to approach the design and programming with this goal in mind; see for example the OO design methods as evidence of this approach to "modelling". However, we believe that great strides in understanding these issues have been made, here and elsewhere, particularly through the notion of constraint programming languages, which are now part of the reality of SE solutions. Constraint programming is providing a powerful connection between computation and declarative modelling, through the idea that computation is the processing of specialized systems of partial information; this is the familiar equation-solving notion, through iterative algorithms "pushing" information.

This abstraction makes it possible to find specialized constraint systems in virtually every arena of computation, involving any kind of data structure. In the discussion, we will more specifically describe an approach coined Model Based Computing, that is now being developed for applications in embedded real time software, and we will illustrate the versatility and power of such approaches to help us build true theories of the world. What is most appealing, in the true engineering tradition, is that it becomes possible to build the software so as to have an implementation, but also to allow monitoring, simulation, control, diagnosis, explanations of the system so developed; tools and methodologies are being developed, exploiting ideas in model-based reasoning in AI.

There is another fundamental concept in SE; that of modularity and of composability of behaviours; this, too, is being revisited, although it is done probably for much more pragmatic

reasons than the first two changes we have just described. With the advent of downsizing, the reality of client/server architectures, the ubiquity of systems connected onto networks, empowering the individual user as well workgroups becomes a priority in the workplace; new demands are placed on programming languages and on the development of applications. We move from a world where the MIS shop was developing applications and delivering them to the end-user, to a world where end-user become power users; they want to move away from the integrated application, and tailor the suite of applications to their taste, to the requirements to get the job done quickly, to do it in reactive ways, depending on the situation in the Workgroup at a given time, depending on the availability of resources on the network, etc. Current programming languages are not languages that are suited to this kind of task, as evidenced by the proliferation of dedicated scripting languages, such as AppleScript, ToolTalk, VisualBasic and others. Although they provide meaningful abstractions to the user, it is clear that the way these languages have been defined, only takes into account simple ways of connecting modules on the network, with little abstraction power and no reasoning capability. It is clear that the requirements will become more stringent, as the usage of these languages become more widespread and move to the end user. What is needed then, is the development of true "glueware" tools, providing the level of mechanisms to deal, beyond the obvious parameter passing, with e.g. synchronisation, co-ordination, identification, localisation and failure analysis requirements. In the discussion we will illustrate this trend which is actively pursued at several places, by the work we conduct on new types of scripting languages, which have roots in powerful modelling technologies and which will bring to the forefront new declarative ways of building distributed and co-operative applications from existing components. Interestingly, it is again the notion of constraint that can provide the abstraction needed for such rich scripting languages. The scripting application needs to be designed without the full knowledge of the environment in which it will be operating; for example, it may depend from the way the network behaves, or from the choices of data representations across the components, or from the behaviours of these components. The requirements of the application are now stated as constraints that need to be maintained, taking into account the state of the component applications.

The above analysis points to problems that need to be addressed to contribute to the future of SE; what we will discuss here is more than enriching the existing; it allows to make strides in contributing to the "productivity" issue in SE. But, because nothing is free, it does so at the expense, to some extent, of generality. Rather, the frameworks we are proposing are general; the instances of the framework will require, however, specific work to be done, to provide the tools and techniques to be used in these frameworks. However, the situation is pretty exciting: the development of appropriate glueware tools that we have mentioned, as well as gaining the understanding about how to open abstractions to make available the information to the glueware, gives us strong reasons to believe that such a custom approach to SE, based on powerful modelling techniques, is feasible.

Acknowledgements: This note owes much to input and discussions with Jean-Marc Andreoli, Gregor Kiczales, Remo Pareschi and Vijay Saraswat.

Alessandro Giacalone

Vision is the primary conduit of information into the human brain.

In terms of support for human reasoning, decision making and action, visualization is one of the most useful, feasible and fundamental functions computer technology can provide.

By visualization one must intend the whole process that takes from sources of data to the delivery of significant information to the individual in visual and comprehensible form. Thus, visualization does not merely mean "user interfaces" as we see them today.

Data and their sources may be of diverse nature. Data may be, for example, numeric, graphical, textual or analog signals, and may originate from repositories e.g. databases or may be generated dynamically, e.g. by simulations or physical processes. Thus, in general the information contents which are relevant for a certain application must be identified and extracted by correlating potentially heterogeneous data. Finally, decisions must be made about suitable and feasible visual paradigms for representing the information and for enabling efficient action on it by individuals.

Visualization techniques have been utilized already for some time in specific application areas e.g. engineering, entertainment, research, traffic control, however, typically at a high development cost and with ad hoc solutions for each single application. One of the challenges is to develop methodologies that render the development of visualization techniques more systematic and efficient/economical, in order to make visualization available for wider classes of applications.

Nevertheless, research will have to focus on introducing visualization techniques in specific application domains. These may be characterized by the involvement of specific combinations of technologies, type and formats of information, or human interaction paradigms. Examples include the following:

Visualization of databases

In addition to visualizing the semantics of data in databases, a question is the visualization of the database itself, that is of the methods for accessing, searching and retrieving information. A particular stress should be placed on the management of heterogeneous information sources. Moreover, as resources will be more and more accessible through networks and telecommunication, solutions will be needed to support the "navigation" and, in general, the awareness individuals have of the environment in which they operate.

Visual information management systems (VIMS)

While related to the previous topic, VIMSs pose whole new challenges, including:

- the sheer quantity of data that need to be stored, retrieved and updated orders of magnitude greater than in the case of alphanumeric data;
- the new problems posed by the indexing and structuring of visual information;
- the feasibility of advanced search methods, based e.g. on extraction of features from images and graphical data.

Virtual and computer augmented realities

In virtual realities, the production of visual information is combined with other technologies in order to reconstruct the environment of operation of an individual completely and with varying degrees of realism. In computer augmented realities, the goal is to enhance the understanding of a real scene by adding useful information to it e.g. water pipes under a street, electrical wires in an engine, medical data about a patient's organ. Thus, the former can be considered a special

case of the latter, in the sense that in virtual realities the real scene is totally omitted. The two areas have in common the fact that the production of visual information must be tightly co-ordinated with the movement and action of the user. This needs the interaction of a number of technologies, including interactive animated graphics, sensors and tracking devices, special interaction devices, audio and tactile feedback and other mechanical devices.

In general, the realization of a visualization system will require the combination and integration of a variety of technologies.

These include:

- Interactive graphics, multi-media and user interface technology.
- Knowledge representation techniques, pattern recognition, inferencing and constraint-based systems.
- Machine vision and image processing.
- Communication and distributed computing.
- High-performance computing.

These technologies have tended to evolve independently, especially in the area of implementation strategies. A challenge is to develop methods for combining them while achieving the performance required by many (most?) end-user solutions.

Brian Gladman

The status and future of the software industry

To judge the future evolution of the software industry it is worth looking at the past and the present position of this industry viewed in the light of experience in a number of more established engineering disciplines such as civil, mechanical, electrical, electronic and aerospace engineering. Engineering disciplines generally proceed through similar phases in their process towards maturity and there are a number of markers which can be used to assess the extent to which this overall process has progressed (this does, of course, assume that software design and development is an engineering discipline). The following paragraphs consider some of these possible measures of maturity.

Design Understanding. Initially design is based on trial and error but as design understanding grows this is replaced by predictive design and development. During this process system and product designs become specialised to particular applications and the 'right' designs become 'community' property rather than those of particular designers or companies. Thus most cars have four wheels etc. Software Engineering still has some way to go here - predictive design is still fraught with difficulty and man large systems (the handling of which is a good test of maturity in itself) still go badly wrong. However, some in some areas (compiler writing, database design etc.) there are now good models so we are clearly on the right road to success. Are we about 5% up the maturity curve?

Component Industry and Infrastructure. Another good sign of maturity is the emergence of component based systems construction and a supporting industry infrastructure (that is a component supply industry). A fully mature discipline will have several industry supply 'layers' and there will be very good, long standing standards, especially simple ones (5v logic, the standard house brick etc.). In the software world the position here is not clear - some standards have emerged (languages, library interfaces, application portability interfaces etc.) but it is not clear that there is a true industry infrastructure which supports component based construction. Some company specialisation is now occurring and there may be the signs of emerging component supply companies with the development of C++ but it is very tentative. On this basis I suspect that we are low on the maturity curve.

The Emergence of Sub-Disciplines. Mature industries exhibit a specialisation of roles and a clear set of relationships in carrying design, development and production forward. Thus in the construction industry there are architects, designers, surveyors and construction companies with a well established set of relationships in carrying a task forward. Such specialisation is less evident in software engineering.

Professional Underwriting of Products and Systems. In mature areas the understanding of the design, development, production and use phases of products and systems is such that suppliers feel able to underwrite the performance of the products which they market. In contrast nearly all software comes with a disclaimer and is often delivered with many faults (this is especially true of mass market software where software testing is completed during the delivery of progressive 'enhancement' releases which are really bug fixes). The software industry seems to have a very long way to go here. Perhaps we are only a little way up the maturity curve on this basis.

How will the software industry evolve?

I suspect we can expect to see developments in the areas discussed. In particular we can expect to see industry specialisation and the emergence of component based construction. Hopefully we will also see improvements in the quality of products as delivered and maybe be even a guarantee that they work as specified.

Companies who have built mass markets on delivering high functionality at low cost will need to deliver quality as well if they are to survive (remember the changes in the TV industry). In

this respect the emergence of a few truly world-wide suppliers, sign of maturity, is happening (e.g. Microsoft).

The emergence of a global information infrastructure may have a big impact on the future shape of the software industry since this may well change geographic factors in a way that favours remote rather than local suppliers. This will also raise complex legal and Intellectual Property Rights (IPR) issues and those of Government control of markets.

The decade from 2000 to 2010 seems about right as the time when the software industry will become substantially mature.

Questions:

- can action be taken to accelerate progress towards maturity?
- what are the best future roles for Government, Industry and Academia in fostering further developments in software engineering?
- to what extent is this an issue which needs to be tackled on a National or International basis?

Software and the global information infrastructure

Until the mid-1980s computer based information systems were typically vertically integrated, stand-alone systems but a change is now taking place with a move to networked distributed systems. The growth of the Internet is a good example of this trend where over the space of a few years a network with millions of hosts and tens if not hundreds of millions of users has emerged. The information services provided on such global networks promise to revolutionise the way we do business.

Global networks and the connected host systems are vulnerable to being attacked by introducing viruses, worms etc. or by attacking the availability of the communications services on which they depend. The Internet is vulnerable in this respect since it is wide open to abuse. In the main it has relied heavily on its users behaving responsibly and, considering the size of the network, this has worked remarkably well with only a few known major attacks on the network as a whole.

However, as such network systems become a part of our business activity, it will be vital to protect their integrity and that of the information which they provide and this will require improvements in the security which such open systems can offer. Software will play a vital role in this since a number of vital algorithm based techniques for providing security services in globally distributed information systems are now emerging. These include cryptographic, hashing and signature algorithms including, for example, public key techniques which are well suited to global networks.

A problem here is that these approaches have had high military value in the past and this has meant that Governments have consciously sought to limit the spread of this technology through export control and related activities. Thus, while open systems approaches have emerged in other areas they are still in their infancy in this domain. Agreement on the methods to be used to protect an information infrastructure which is global in scope thus remains a thorny political problem.

The emergence of knowledge of information security techniques which can be implemented purely in software is likely to have a big impact here since it is no longer clear that Governments can (or should) limit the spread of this technology.

However this has not stopped them making such attempts and the United States is trying to use the dominant position of its major software supply companies to wage 'information war' on the rest of the world by selling 'weak' commercial products overseas (0 bit encryption) whilst deploying stronger technology at home (but which allows the Government to 'tap in' under legal controls; that is Clipper). None of this is consistent with a global information infrastructure offering good levels of security sufficient to support business and commercial applications.

This is an important area of policy development in which the actions of Governments can have a major impact on the development of new markets in the software domain. It is worth further consideration during the Software 2000 programme.

Questions:

- can a truly global information system infrastructure develop without international standards for the provision of information security?
- is there an acceptable mid-course between those groups who wish to see this technology widely deployed and those who wish to constrain it?
- given that mathematical algorithms and source code are involved in much of this work, are Government constraints feasible or acceptable?
- given the international nature of the problem, where should decisions be taken on to balance between wider deployment and continuing constraints? Given the strength of market forces, will deployment occur anyway?

Andrew Herbert

Whither standards

In some respects standards are helping - e.g. all databases now talk SQL. In other respects standards aren't helping - the various bodies standardizing UNIX seem mostly to have added further variants to the species rather than reducing them. The computer industry is losing interest in the ISO process as being too slow and divorced from business needs. Industry groups such as OSF and OMG have made rapid progress but seem to run out of steam once they've reached first base, or else they become tainted as vendor camps in competition with one another. Users have little confidence in standards, except where major industry sectors can gang up and form purchasing cartels, such as POSC.

How is all of this going to develop? Will we see more user cartels? Will we see vendor alliances replacing the de facto and de jure standards process? Will we see a small number of "killer products" licensed by everyone providing the base technology?

Can mainframe and UNIX pricing strategies withstand the onslaught of cheap PC software?

UNIX software costs ten times as much as the equivalent PC package. Mainframes often cost another factor of ten. Are there benefits that justify these differences?

Information highways (to pick a phrase of the moment)

What impact will these have on software? How much data integration and applications building will be done by the information providers? Will the domestic unit be simply a "terminal" or will it be a real computer running software purchased by the user? Will software be purchased over the network? How will accounting be done, especially between information providers - e.g. who pays who what if I put your cookery recipes in my on-line "Lifestyle" magazine. I guess this strays into copyright issues, but also begs questions about policing, charging, licensing and so forth.

Who writes programs anyway?

In the database world, and to some extend the GUI world people don't write a great deal of software any more. They use tools, often graphical, to express high level requirements declaratively and application generators do the rest. Some people report as much as 80% of the code of a new application being automatically generated. This approach is attractive since it leads to robust code quickly; often the performance hit is not too bad. Will we see the same capability spread out into other areas such distributed computing? Who will be the tool vendors: start-ups? Software houses? Systems integrators? Most tool sets are oriented towards the single user, or a programming team and depend upon some sort of database at the lowest level and there are few interworking standards at this level. Will it be possible to interconnect design environments?

Mobility

What kind of demands will mobile electronic gadgets but on software. Will they demand simplified operating systems and GUI packages, or will the size of machine you can stick on a small low power unit be big enough to run monsters like the DCE and MOTIF, or NT and Microsoft Word?

What are the killer application and their software needs?

In the area of distributed computing the current focus seems to be on "integrating applications" for utility companies ('phone, power etc.) which provide customer service agents with a uniform way of accessing the utilities databases and checking on the progress of activities flowing through the organization. It involves building a "horizontal" structure across a typically "vertical" set of applications. This is a tough job involving networking, heterogeneity, data consistency, data replication and real-time response. Solutions are ad hoc and built ground up. This takes too long and costs too much. How will the software industry respond to this challenge?

Cliff Jones

The "issues" that I would like to see discussed at Hedsor are the following:

- Information nets (see position paper below)
- Education (experts versus IT users)
- The role of "formal methods"

Information nets

Those participants at this workshop who know me would expect me to present a position paper related to `formal methods' (and I am happy to discuss this topic at any length!) but I think there is another topic which interests me and is perhaps more suitable for SW2000 discussion. Let me start with an anecdote. I worked for many years for IBM and in the mid 1970s I remember vividly a discussion with one of that company's senior executives. Basically he was talking about a vision of the future in which companies might provide "computation grids". The analogy was with the National Power Grid which provides electricity to our homes and the idea was that we would be able to plug into this grid in order to do computation. I saw this as completely irrelevant to the future of IT. What I wanted to see was an information grid, something which would enable the ordinary person to access and possibly manipulate information which was important in order to make their lives either more productive or more enjoyable. The reaction from the executive was amusing - that project would be too expensive for a company like ours!

We're in a situation where there is an enormous mass of data available in machine readable form. We are now moving to a situation where we can begin to access some of this data over World Wide Web, Gopher, etc. but this data is not information, it is just not structured in a way which makes it usable. Furthermore there is very little opportunity to manipulate or connect other pieces of information into the data which is accessed as ASCII files. I am convinced that information technology for `real people' is about access to information (not computation per se - but computation will be required in order to manipulate the information into the form that the user desires).

Information is structured data, not strings of bytes. (I don't want to go into details here of what formats are or are not desirable but something like a collection of entities and relations seems to me a reasonable working hypothesis.) To a first approximation there is zero useful information in this sense available to me at my workstation today. If I list the tasks I wish to undertake either in my professional work or to support my outside interests I find that the tools available to me reduce to things like 'grep'ping a large series of locally stored ASCII files and limited access to locally designed databases. What I want to do is to access data stored and maintained on other machines and to be able to connect information of my own into those databases so that when I next look at it I can be reminded of my previous interaction with that data.

Again, let me report an anecdote - this time a recent one. We were recently approached by an engineering firm who reported that they needed help with database technology. During the meeting that we set up in order to understand their problem, it became clear that their task was indeed access to information. The firm is involved in designing water treatment and sewerage plants. They need information about population and trends in the change of population; they need information about the terrain in which the potential storage and processing facilities could be built; they need information about the weather and the way in which rainfall correlates with geographic information about rivers and their rates of flow; they need information about the state of pollution in those rivers in order to plan their processing plants. The good news is that much of this information is available in computer databases; the challenge they face is to access these totally separately designed databases and correlate all of the information into a coherent picture. In fact I formed the impression that these water processing engineers were actually turning themselves more into database interface experts than physical engineers. There is for them a huge economic advantage to be able to access and process information rapidly.

This suggested issue that we could discuss at SW2000 comes not from recent painful experiences alone. John Gurd and I are trying to write a paper `Making Sense of Information' for a volume which attempts to set the future research agenda for computer science. We have in fact found relatively little recent material which relates to our concerns. The paper by Mario Tokoro on `Computational Field Model' and his more recent `The Society of Objects' are some of the few recent publications which seem to reflect our concerns. If nothing else a discussion of this topic at Software 2000 might point me to literature of which I am unaware.

Michael Lesk

I would like to see the following points discussed:

- Code retrieval and code libraries: how can one find an already written program that does what you want? I doubt "archie" is the best possible answer.
- What is the future of software for information management? If the "information superhighway" comes to pass, what needs to be done in software to make it effective -- image, sound, text processing? Will we discover a sea-change in programming style as we basically switch from calculating answers to looking them up?
- Why is software so hard to change? Why do we use software that is older than hardware? What can we do to make software more modular and to make it easier to swap pieces in and out?
- How can we apply graphics techniques not just to displaying the output of programs, but to help write them? Why do we still handle code as if all we had were ASCII terminals?
- Why is it so hard to define what a program should do? Do we need machine-readable specifications, software to generate documentation, or what? If we did learn how to define what programs do, would we then see a vast transfer of software work to less developed countries with lower labour costs?
- Why are some of the favourite research topics of the last twenty years (logic programming, artificial intelligence) so slow to move into the commercial marketplace and produce useful results? Are they basically not going to be effective on practical problems, or do we have a problem in the industry adopting new techniques?
- Legal problems: Intellectual property issues and strict liability issues [For the non-US: broader and broader forms of intellectual property protection are being accepted by the U S legal system, including copyright on the "look and feel" of a computer system and patents which take forever to issue and sometimes seem very broad; also, the courts may impose a "strict liability" standard on electronic information publishers even though it is not imposed on paper publishers].

Jessica Litman

How much intellectual property protection for software is best?

Have we been overprotective of intellectual property rights in software and other computer-related works? Is intellectual property law hindering rather than promoting innovation? What aspects of software works should the law leave unprotected?

Are today's industry leaders using intellectual property rights to hinder tomorrow's competitors?

How can we draw the line between offering enough intellectual property protection to encourage the development of software, and offering too much protection, so that intellectual property owners can use the laws to discourage innovation and competition? Do elaborate procedures (like those involved in "clean room" software design) to avoid liability make any sense from a software designer's standpoint? Should we enforce contracts that give software proprietors more extensive intellectual property rights than the laws do?

Are legal standards too unpredictable?

There is always a trade-off between laws that are predictable and certain, but that may be too rigid to respond appropriately to changing technology, and laws that are flexible and therefore uncertain.

How broadly should we privilege reverse engineering

Should the law limit any reverse engineering privilege to particular purposes (such as interoperability)? If so, how should we define them? Once one has reverse engineered a program in order to achieve interoperability, can one use what one has learned for any purpose so long as the software one creates does not itself infringe the software one reverse engineered? Should decompilation be treated differently from other reverse engineering methods?

Interoperability

How should we define interoperability? Do the policy reasons that support the development and adoption of standards also support a broad definition of interoperability?

Software protection and the "Information Superhighway"

Do we need new legal models to extend intellectual property protection, or to enforce intellectual property rights, in an era in which distribution of new works is more likely to occur through electronic transmissions than through the purchase or lease of tangible copies of protected works?

Bill O'Riordan

One can harness the power of a horse to pull a cart up a hill but what about using one hundred and twenty eight chickens? This problem illustrates both the promise and the difficulties of parallel processing - if one solves the problems associated with the control, interconnect and usage of the great range of cheap RISC and CISC to do the job previously done by single powerful and expensive systems almost unlimited cost effective transparent computing power will become a reality in the World of Open Systems.

Initially the portability problem must be solved if parallel processing is ever to become more than a niche market for specialised applications. There has recently been some exciting progress in this area and there are now several attempts to construct truly general-purpose parallel computers under way but few will succeed unless they recognise their dependence on software.

There are also other important issues, however, that are likely to be increasingly important for programs running on complex parallel systems:

Verification

This is the question of program verification and reliability and is obviously especially relevant to Large Databases and safety-critical applications.

The verification problem is with us for programs running on serial computers: on parallel machines there are a number of complex additional surmountable problems that confront the programmer which will task our intellectual competency and is something we cannot buy from any other vendor. (This is the subject of discussion of Algorithms and Complexity.)

A locally parallel machine will import into a local computing environment or even a workstation the multi system organisational problems that are well recognised in larger scale systems such as networked systems and distributed databases. This level of verification will now need to be extended to this area and may well ensure that only vendors with existing experience in the area will succeed in moving credible and reliable products. ICL can confidently handle this.

Non-determinacy

Non-deterministic behaviour of parallel programs is not restricted to distributed memory multiprocessor systems. In a detailed study of parallel FORTRAN programs running on shared-memory multiprocessors, McGraw and Axeirod concluded.

"The behaviour of even quite short parallel programs can be astonishingly complex. The fact that a program functions correctly once, or even one hundred times, with some particular set of input values, is no guarantee that it will not fail tomorrow with the same input". Without some formal tools, tracking down bugs in parallel programs ran therefore be exceedingly difficult. This is achievable within ICL because of its experience in large integrated systems.

Deadlock

One of the most common parallel programming errors is that of deadlock - all processors are sitting waiting on an input from another processor. In constructing parallel programs for Multiprocessor hardware there is a great need to concentrate on deadlock free methodologies. We will need to be very selective in this endeavour.

Issues in multiprocessor architectures

This presents for ICL the most exciting opportunity of all. I have heard it described by some as an "insurmountable opportunity" but what has emerged is that we are probably capable of

being the first major I.T. Company capable of resolving it as we have recognised the problem earlier than our competitors.

The issue is how to organise Multiple processors and their meaning into a "coherent system". There are two main camps: shared memory, in which multiple processors, each typically with some local cache, draw from a common high-speed memory, and distributed memory, in which each processor has its own private, complete memory. The argument is usually expressed in highly technical terms, with architects, designers and development Engineers and programmers of each machine type adamantly proclaiming its merits, with total disregard for what the end user wants - which is to service his complex application requirement in the most effective and competitive way possible.

Both sides because of historical product evolution are dependent on the primitive state of today's machine-specific multiprocessor software, which forces them to write code for a specific architecture at or near the machine level.

Programs for shared memory machines are therefore written with UNIX threads, with shared data simply dumped into common memory and arbitrated with semaphores, spinlocks and other machine-specific calls. Programs for distributed memory machines are typically written in a message passing formalism, in which near-complete programs running on each node communicate explicitly with one another through point-to-point messages. The massive amount of time invested by designers and development programmers in mastering these primitive systems, combined with the near-absence of commercial applications for end users, means that the argument is dominated by the architects, designers and development programmers, with rigid opinions formed by painful firsthand experience. (CERN Met Office and UNIVERSITY engineering departments etc. are good examples).

In the commercial computer world, the arguments tend to be conducted in the rich, well-developed tools and applications. The closest analogy here would be to imagine the RISC/CISC debate if compilers were not yet invented. RISC proponents would be arguing that their instruction set was easier to program because programmers needed to memorise fewer assembly mnemonics, while CISC supporters would claim that their richer instruction set shortened the length of a typical assembly program.

The existence of applications written in C, a high-level portable syntax which has all but obviated assembly code, means the RISC/CISC argument goes on anyway, but on the much more practical and relevant plane of what applications are available, and how fast they perform on each architecture. C and other high-level languages allow our customers to participate meaningfully in the discussion (and buy product based on this participation), rather than watching from the side-lines as the technologists etc. argue their cases. The very good news for ICL and the shared/distributed argument is that its equivalents of the C compiler in RISC/CISC have already been invented, but are only now reaching a broad market, more than ten years after the introduction of parallel computing hardware.

I have come across these at the LONDON Parallel Application Centre and Imperial College. Several programming paradigms work reasonably well on shared or distributed memory machines, including C-Linda, and its public domain version from our friends at Edinburgh University, Strand 88 (Prof. John Florentin) from Strand Software, and Express from Parasoft. In addition, some next-generation operating systems, such as CMU's Mach, and possibly CHORUS will offer low-level facilities which can hide substantial differences in memory architecture. Object oriented techniques e.g. data encapsulation, lend themselves readily to automatic analysis for parallel execution. C++ in particular, is supported by numerous hardware platforms and is reasonably known in the commercial development community. ICL in association with University College London is collaborating in the development of a software environment providing parallel execution of suitable C++ programs without any need for the programmer to consider details such as the memory architecture or interprocess communications.

With well-designed tools, even developers need not know anything about the memory architecture of the system they are using, any more than C programmers need to know the assembly language of their CPU. The end user is even further removed from these concerns. Now that good tools and operating systems are available and machine-independent

applications are finally being delivered, users are getting their chance to compare products based on criteria that really matter: what applications are available, and how do they perform?

Multi-processor performance is often driven by data availability, measured in latency (how long the first chunk of data takes to start arriving) and throughput (the sustained speed at which entire streams of data can arrive). Not surprisingly, combinations of shared and distributed memory can probably do the best job of keeping local data fast and local, and distributed data widely available. Some early signs of this convergence are the new burst transfer modes of FutureBus, VME64, Micro Channel, and nuBus9O, which bear an eerie resemblance to message passing; the increasing use of HIPPI to integrate distributed-memory clusters of several shared-memory machines; and the inclusion of broadcast protocols to stimulate shared memory in the newer distributed-memory architectures. This latest software which I see, provides what best might be described as a "don't Care layer" so that developers and users can get on with things. After all the arguments are over, it seems that most successful systems will be shared/distributed hybrids, and users may never know the difference.

Application match

This is perhaps the most important issue in determining the success of parallel computer architectures because what will make them acceptable to a general (NON-niche) market is their credible performance of useful applications rather than what goes into a particular box. Selling architecture is rather like selling safety in car's - it won't work. It must be within a credible framework.

ICL is fortunate in having controllable access to a technology that makes the exploitation of any potential parallelism in an application an automatic and more importantly a transparent process as far as the application designer is concerned. The technology, ELIPSYS/ECLIPSE (from ECRC and Imperial College) has the added value that it is still only in its first stage and can be expected to demonstrate substantial further increases in performance when 'and' parallelism is exploited.

Without an application building tool that automatically exploits parallelism, the underlying platforms will continue to be relegated to niche markets that require scarce programming skills not generally available or acceptable to the general mid-range and below-systems market for IT.

Brian Randell

Lessons from history?

I have long had a strong interest in the history of computing, though my historical researches have concentrated on the origins of computers, rather than their more recent development. In recent years I have had little time to pursue this interest, but two events have driven me to think more carefully about the more recent past, and to wonder what lessons can be drawn from it.

The first of these events was an invitation, which I was too flattered to turn down, to rewrite the final section of that splendid lavishly-illustrated history, "A Computer Perspective", which had been out of print since soon after it was originally published in 1973. My task was to replace a four page account of the period 1950-1970 with an account of the period 1950-1990 - but still in just four pages. The second event was an invitation from the IEE to give a lecture at their planned Charles Babbage Bi-Centenary Conference, summarizing the entire development of programming from Babbage to modern times. This also I accepted, though I should have turned it down, if only because the conference was cancelled at the last moment.

The result, however, was that I was forced to think very hard about the history of the last forty years or so, and about what if anything we can learn from it. It was a sobering experience when I reluctantly reached the decision that, despite all the software research and development that has been undertaken, the two most important events in the history of software since the invention of the stored program concept were the development of (i) FORTRAN, and (ii) the microcomputer! (The former established the recognition that it was practicable to program at a level which was significantly more abstract than the machine code level; the latter caused what was in effect a huge programmer and user population explosion. This led to such a great increase in the number and variety of creative new applications, and also to so much production and marketing of lots of essentially similar software systems that the normal processes of competition and evolution at last started to work fairly effectively.)

Between these two events, I regard the late 1960s as somewhat of a watershed. Not only was this when IBM unbundled its software, so creating the first real possibility of a software market and indeed an industry. This was when two NATO Conferences identified what was termed "the software crisis" and first promoted the concept of, and the need for, a body of knowledge and practice worthy of the term "software engineering". It was also the time when the concepts of software components and software re-use began to be talked about seriously, and when the technology that is currently touted as central to these concepts, namely "object-oriented programming" had its real origins, in SIMULA-67. However to my mind the first widely successful use of software componentry occurred shortly afterwards. This was the application of a much simpler scheme, namely the pipes and filters of UNIX, to the valuable but special case of applications which could be cast in terms of operations on linear character streams.

The software design task centres on the problem of mastering complexity - the new complexity that each new application provides. The classic (perhaps the only) technique for dealing with complexity is of course "divide and conquer" Some developments have, so to speak, divided off part of the complexity, and provided a canned solution of general utility (e.g. to the problems of dealing with physical storage, or the fact that the application is being carried out on a set of computers rather than a single one). This sort of development can often be employed quite readily, and so will be taken up relatively quickly. But only gradually have various commonly useful standard facilities been identified, and provided in this way.

Other developments are more related to the actual division process. For example UNIX pipes and filters, or object-oriented programming for that matter, really just provide a convenient form of mortar, with which the programmer can glue together whatever bricks he or she chooses to invent or obtain. But a new form of mortar does not help one to invent an improved set of bricks - the subdivision task (equivalently, the task of *choosing* interfaces, or of *inventing* languages) remains and still requires creativity (or at least an ability to recognize that some pre-existing component would suffice). Neither for that matter does a new, even completely formal, method of specifying such interfaces or languages directly aid the task of choosing just what is

to be specified. Thus creativity of course continues to play a pivotal role in the software development process.

Creativity is difficult to inculcate, to improve significantly, or to schedule (leave alone to automate) and varies greatly between individuals. This is as true now as it was twenty-five years ago. Thus it is perhaps not surprising that, to reiterate my earlier point, it is my regretful conclusion that the last twenty-five years of research and development into software engineering, programming methodology and the like, though they have produced much of merit, have not had as significant an impact as an event which is quite external to them, namely the advent of the microcomputer, which caused such an increase in the number of people involved in the creative process.

It is always dangerous to use the lessons of the past in an attempt to predict the future - but they are probably the best basis available. I am reasonably sure that software engineering, and in particular software componentry and reuse, will develop and be aided by the various efforts now going into object-oriented design tools, etc. But progress in establishing effective and widely-usable object libraries will I fear be much slower than most people hope and expect - because of the creativity required. Indeed my strong inclination is to predict that the next ten years, say, will again see the software world affected more by other external events (some of which one can already see starting to happen, or looming on the horizon), or by some unexpected new idea, than by its general research and development into software engineering and programming methodology.

The past as millstone

The past is not only a source, albeit of dubious trustworthiness, of guidance as to the likely future, it is also - especially in the case of software - an increasing source of impediments to change, if not to progress. (Such impediments have in recent years being engagingly termed "legacy systems".) Most serious software cannot be designed for a "green field site". It has to support old interfaces, old data formats, old methods of working, etc., as well as provide some improvements. Only by such means can it fit in alongside existing investments in hardware, data, software and people, and produce an overall economic benefit.

The speed with which new developments will be taken up will thus often depend in large part on the effectiveness with which they have been designed to cope with the past. A striking example of success in this regard is World Wide Web - one of the most elegant aspects of its design, contributing greatly to its rapid spread, is the way it enables continued use of a variety of existing systems such as Gopher, FTP, WAIS, etc. And the operating system battles now raging will evidently be won in part by demonstrating successful support of existing applications.

Living with the past is even more complicated when the problem is to introduce new software into a continuously running system. This problem has been faced most severely by the telephone companies - their level of success is remarkable, but by no means total, as recent major system failures have shown.

Predicting the effect of these problems on the various segments of the software industry is not easy. Logically, one might argue that legacy systems are by definition an ever-growing impediment, and hence will slow down development more and more. But who knows what novel techniques for coping with such systems might be developed, and prove so effective that the millstone proves ineffective. The very invisibility of software is a great aid here. One can glue ill-fitting systems and components together by a byzantine network of emulators, converters, filters, etc., and rely on computer power to reduce their performance impact to a level which is acceptable. (The fact that the equivalent design represented in visible hardware components would probably arouse ridicule is perhaps best ignored!)

The characteristics of software

The most fundamental characteristics of software over and above the fact that it typically embodies immense complexity, are (i) that it is intensely brittle, and (ii) once developed, it can be replicated at virtually zero cost. The brittleness is of course logical in nature, and comes from its digital character. No recourse can be made to normal notions of continuity and of safety factors in designing the software, in validating it, in estimating its dependability, etc. Research on probabilistic algorithms, and on inherently self-checking programs, could be regarded as embodying a slight move away from this situation, but logical brittleness is likely to remain a major characteristic of software, and continue to impact our ways of producing and using it. I foresee at best only slow incremental improvement in our ability to deal with this characteristic - and find it difficult to imagine developments external to the software world which will have much impact on this situation.

The extremely low replication cost comes from the technologies, all digital in nature, that we have for embodying software - technologies that are being increasingly used for many other sorts of information. The use of these technologies provides a set of commercial and legal problems (termed the problems of "digitized property" by John Perry Barlow) to all involved in any form of information storage and transfer, from libraries to film producers, from book publishers to software houses.

Developments in networking, in security mechanisms, in copyright law, in patenting practices, etc., could be equally relevant to all forms of digitally-encoded information, and could have a major impact - not least on the various components of the software industry. What these developments are likely to be is something I leave to people more expert in the various fields to predict - I content myself by predicting that one or more of these developments is likely to be the major determinant of the future of the software industry.

Of course, aside from these general characteristics, one can identify a number of characteristics which effectively segment the software field. For example, the problems of producing safety-critical software for an individual process control application are very different from those of producing software for embedding in some mass-produced household appliance, which are in turn very different from those of successfully competing in the shrink-wrap software market. The relative seriousness of the various Issues that I have discussed, and hence of predicting their impact on future developments, will vary across these different segments. But discussion of this point goes beyond the intended scope of this brief position paper.

Concluding remarks

By way of concluding remarks, let me simply use the following quotation:

> The future is dark, the present burdensome. Only the past, dead and finished, bears contemplation. Those who look upon it have survived it; they are its product and its victors. No wonder therefore that men concern themselves with history.
>
> G.R. Elton. *The Practice of History* (1967)

This quotation is one that I have used once before. This was in 1978 in an Invited Paper, at the International Conference on Software Engineering, in which I surveyed developments since the NATO Conference ten years earlier which had first launched the Software Engineering bandwagon. In fact, though I very much like this quotation, and felt that it had a similar potential relevance to our workshop and so could not forbear from including it here, I actually did not and do not subscribe to the sentiments it expresses. I study history because I find it enjoyable, not as an escape from the future - which in fact I find even more fascinating to contemplate.

Gill Ringland

Three observations to start:

- The IT industry as a whole is changing its centre of gravity. Gartner Group point out that mature industries normally reflect 1/3 expenditure in the business sector and 2/3 in the consumer sector. This spend does not necessarily reflect through to the proportion of software value, but certainly has implications for the style of provision. For instance, will consumer acceptable interfaces redefine business interfaces? - what is the diffusion path? What are the commercial structures that support the consumer market? What knowledge structures?

- The software business is still "unstable" in the sense that there are no strong business parallels. The software industry is still "off the chart", in the same way that the price umbrella of IBM kept the IT industry "off the chart" on most business parameters (using for instance the PIMS database of several thousand companies and divisions of companies) until the last few years. Weak matches for software businesses are found with films, and book publishing. What model should we anticipate for the "end point" and when should we expect this to be approached? Or is software the first of a new style of business?

- The reliability of items for the consumer sector is defined in terms of (e.g. for washing machines) "1 in 3 are expected to break in the first five years". What is the equivalent specification for software end to end, and what are the implications for components? Given the trend of the industry to "horizontalization" with "a few" suppliers of each software category world-wide, how can these suppliers usefully model the cost benefit of their engineering - what tools and techniques do we have for this?

These suggests that topics for discussion should include:

- what are the paradigms for HCI in the year 2000 and beyond, and what are the cultural (multi-cultural) and knowledge-span factors behind the HCI?

- what is the business model for the creation and trading of software components and what will cause discontinuities?

- what tools and technologies do we need for specifying and authenticating software components in the absence of a close-in user model?

- what type of person should we be looking for to build software and how can they be educated so as to develop as the business evolves?

The environment for software

1. Introduction

This position paper is based on work done as part of the ICL planning cycle. However, since many of the assumptions identified here underpinned the discussions at Hedsor, it is included as a post-Workshop position paper.

The economic, political and social factors in which we operate are changing - and predictions of the future are liable to be wrong in at least some aspects.

However there are a number of major predicted trends and discontinuities which have significant implications for us.

We have discussed the range of potential scenarios in detail with Stanford Research Institute and Henley Centre for Forecasting, and have incorporated their comments in most cases. In the remainder, where they emphasised the importance of technological developments, we were more cautious about the ability and take-up rate for these to be absorbed into the business and home environment.

Similarly, we incorporated the view of external consultants that the rate of change in the IT industry will continue to put brutal requirements on management.

We have incorporated views of the IT industry from Gartner, IDC and McKinsey who expect:

- the move of the industry to "a few" majors with global reach,
- that the size of the IT industry is now large enough for it to be impacted by overall GDP performance.

2. Economic factors

The economic picture for the next few years shows lower growth of GDP/capita in the G7 countries than was seen in the 1980's. High growth of GDP/capita is seen in SE Asia, Latin America, and in "city states" in China and India. The differences in growth rate are such that the SE Asia block is expected to approach if not exceed Europe in purchasing power early in the next century.

European growth is forecast to be lower than that of Japan and North America, and Credit-Suisse First Boston believe that there is a possibility that Europe might tip into negative growth.

There are signs that the late 1990's may see a surge of growth: that networking and multi-media technological innovations could fuel the creation of new businesses with an energy similar to other major industrial sea-changes. This is likely to start in North America, and current activity is focused around information services (e.g. home shopping, entertainment) and collaborations between film/video/cable suppliers and IT companies.

The financial impact of the ageing of the population will be offset by the reduction of school/university people - there will not be a significant net drain on the exchequers until after 2000 in most of European countries or Japan. Unemployment is a financial burden, particularly in Europe, through the decade, and the costs of the elderly (pensions and particularly health) increase.

2.1 Implication for the IT Industry:

- the purchasing power of the NICs, in business and consumer markets, represents a business opportunity as well as an option for offshore manufacturing.
- with low growth in the G7 countries, the business market will tend to a replacement market with decreasing price for the same functionality.
- a spate of joint ventures and partnerships across industry boundaries to explore changes in technology which will unlock markets - e.g. retailing, home education/entertainment.
- IT opportunities for growth will also be found in meeting major economic changes e.g. in the demographic shift to an ageing population with requirements for leisure, travel and health care.

3. Political framework

It is widely assumed that there will be three trade blocks - North America (possibly with NAFTA), Europe (with some of the EFTA countries and links to some of the Eastern European countries) and Japan.

It is expected that North America will keep pushing for free trade, that Japan will increasingly regard it as in its interest, and that Europe will not erect tariff barriers against North America and Japan.

In terms of the EU progress, the assumption is that the countries will maintain local culture but that currency fluctuations will not be a barrier to trade inside Europe by Year 2000.

The IT industry in Europe will continue to become more global, with the former local IT champions becoming parts of global groupings (e.g. IBM/NEC/Bull in France, DEC/Olivetti in Italy, Fujitsu/ICL in UK). There is no barrier to North American services suppliers in Europe.

Europe is deregulating & privatising fast. This applies not just in telecoms, where the deregulation picture has changed since earlier this year with cross-border satellite & cable companies forcing the pace, but also in utilities, airlines, etc. This leads to decreasing control at national level and increasing crossborder trade.

We build into our working assumptions that there will not be a major war in Europe (though increasingly a danger of "small scale" wars, and use of the stockpile of nuclear weapons from former USSR), or a war in the Middle East which causes an oil shock.

3.1 Implication for the IT Industry:

- the IT industry's technology focus will continue to be the US, and the taste and experience lag between Europe and Japan, and North America, will cause architectures, commercial practice and technologies to be set by de facto standards in North America.
- opportunities in Europe will include major growth in deregulated industries, particularly exploiting telecoms capability.

4. Social trends

The role of the large organisation world-wide is expected to change, employing fewer core staff for a shorter "standard" working life, with most people working for smaller specialist companies in association with the majors with lower job security. This pattern has existed for at least the last two decades in Japan and North America, but in Europe it goes against cultural attitudes and EEC social legislation. However mobility of work and leisure will increase in European business over the period.

Immigration into North America will continue to create a multiethnic population with high population growth and labour flexibility. Immigration from SE Asia and Latin America will create strong trading links in due course.

Decreasing relative wealth in Europe is expected to lead to discontent, and may focus on pressure against immigration from North Africa and Eastern Europe as a symbol.

It is not expected that environmental pressures will cause major behaviour changes over the time period in Europe, though the trends already seen in North America and Japan will continue.

It seems possible that significant differences are emerging, between North America and Japan on the one hand, and Europe on the other hand, related to the attitude to innovation including the ability to exploit IT.

4.1 Implication for the IT Industry:

- the technology (networks, mobiles, desktops, information services) to support the new "associated" organisation will be important in North America and Japan, and increasingly in Europe.
- the provision of effective networks for business traffic in Europe, especially across national boundaries, may be slower than projections based on US experience would imply.
- "legacy" staff skills in Europe, Japan and North America limits the ability of the existing IT players to adapt to new opportunities and shaken out staff depress the markets for a number of IT related services (customer service, professional service).
- IT applications/HCI which breach the technophobe barrier will create new markets.

5. Technology trends

It is widely expected that the pace of technological innovation will continue, giving price/performance curves continuing on their current trends into the next century. Mainframe and server prices will particularly be affected. This will impact the IT industry, in that many IT

organisations which have operated in previous price/performance regimes will have inadequate skills for dealing with the pace of change.

The effect of changes in hardware reliability on customer service revenues and company profits has also had a major effect on the business model for the industry.

There is a rapid approach to saturation penetration of desktops in North America, Japan and Europe. Replacement markets will be driven by new applications, and the marketing and support structures for these are different from the traditional "IT skills", being related to user applications whether in business, education or other fields.

The IT industry is expected to continue its divergence into customer facing and product (distributor facing) organizations.

The shape of industry is also changing with new competitors - especially the telcos, cable/satellite/information/games companies packaging IT services differently, mostly for home but also into business. These competitors have the cash to run "new" businesses at a loss to enter new markets.

The penetration of cable TV and other high bandwidth connections to the home is expected to slow the growth of home (multi-media) based services. Satellite TV, particularly outside the US, presents an opportunity for semi-interactive applications.

5.1 Implications for the IT Industry

- the traditional players may shrink even faster than the market if newer players have the right skills for the new opportunities.
- there will be downward pressure on prices for services because of the decreasing requirement for customer service, the costs of taking staff off the payroll, and so contracts are bid for at marginal costings.
- the I.T. industry is looking for new applications of IT, and new entrants from neighbouring markets (entertainment, instrumentation, publishing) may provide them.

Charles Simonyi

Issues

General

Software patents: should we have them? How to educate legislators, courts? What are the benefits and dangers?

Information Superhighway: incredible software market, technical problem, business opportunity, distribution tool.

Ubiquitous computing (wallet computing): real objects (such as a suit in a store, or the policeman who stopped you) can interact with computations and your personal wallet computer (also called inverted virtual reality)

Specific issues in my area of interest:

Software Immortality: Literature and music have been immortal for some time; with digital recording musical performances have also become immortal. Software effort is encoded with reference to so many decaying artefacts that its lifetime is often shorter than its validity. How can we make legacy code immortal? Should we?

What was wrong with McIlroy's 1968 dream of software components? This is not a rhetorical question, but a real one.

Is current software inherently unshareable or are we just not doing a good enough job? Are program generators the key to reuse? Is partial evaluation the key?

Where will European software innovation come from?

How can we make the following widely available, and should we?

- transformational programming
- innovation in abstraction (currently innovation occurs only in new languages which is at best a zero sum game)
- DWIM / other Lisp tools

Position statement

I enclose a Position Statement I wrote in 1989 for the NRC Complex Software Workshop. Software Engineering moves very slowly so I feel fine about reusing the note with only minor edit. One offshoot of the paper was that subsequently I got green light from Microsoft to focus on the subject within the then, newly formed Microsoft Research organization, which is led by Rick Rashid.

We have a team of researchers and programmers working on a project called "Intentional Programming" (IP). The goal of the project is to create programming tools which will be used first internally to Microsoft, and later released as a product. We have a few ideas which we consider proprietary and which we do not yet talk about. This unfortunately prevents me from making sweeping overviews of the total system. However, I can say a few things about our views on which of the generally considered problems and solutions are the key ones.

We think software reuse is critically important - this is hardly controversial - Our experience has been that reuse has remained superficial even when "not invented here" has been eliminated (by moving personnel), incentives have been employed (for example, by stock options), reuse opportunities have been identified (by technically savvy management), and object-oriented techniques have, been employed (using C++,) What remains after all this is an inherent

mismatch of goals between the producer and the consumer of a shared artefact. The producer wants it to be general so that the market will be sufficiently large to support the effort, while the consumer wants something highly specific - so that the performance and feature set will he comparable to what is needed and what could be produced separately. The fact that the producer and the consumer are within the same organizational umbrella does not really change the picture under the following conditions that are normal in our industry:

- the shareable artefacts are relatively small - the larger the artefact the larger the variance in the requirements - often non-linearly larger. (for example, due to the combinations of subsystems implementation possibilities). Even for small artefacts, the requirements are seldom if ever identical (for example, in the implementation details of parameters).
- general solutions (solutions lacking in domain specific knowledge) perform worse and often non-linearly worse than special solutions.
- communication costs between programmers are very high (especially with the smaller artefact) and can be non-linearly high if trade-offs have to be negotiated.

Intentional programming addresses each of these phenomena:

- implementation detail is separated from computational intent (hence the name "Intentional" programming) so we have the benefits of program generators but without their costs; variances in requirements become parameterized invariance.
- domain specific knowledge can be added routinely using program transformations.
- trade-offs are simply eliminated by providing great abstraction capabilities (to express the commonality) and similarly powerful specialization capabilities (to express the specific needs without run-time costs.) The latter is achieved using partial evaluation of the general code.

It is also important to note that under intentional programming, the concept of shareable artefact includes not only subroutines but also transformations which would be typically performed manually by programmers in each instance. For example a storage allocator library routine is a normal subroutine in C. However, the act of updating a program so that an array is moved from static storage into allocated storage is not typically considered an artefact, it is just work that programmers do. Under IP the updates (transformations) would be part of the package together with the allocation subroutines. This means that we not only want to improve the chances of reuse for any given artefact, we also want to make more of the programmer's contribution into potentially reusable artefacts.

The basic system does not involve any artificial intelligence. Our goal is not to simplify the "thinking" part of the programmer's job, but to help the programmer with the more mechanical aspects of programming. For example, the programmer has to make the decision about what implementation method should be used, or even whether the pre-conditions for a restricted implementation exist or not. The system will, in turn, apply the transformations that are required. This has been called by some "semi-automatic programming". The programmer's work will not get simpler, in fact programmers will have to think more per hour. In return the work will be much more reusable.

Complex software systems workshop

What do you consider to be the worst problem, you have with current software production, and what suggestions do you have for alleviating it?

The worst problem affecting my activity is insufficient productivity. In the microcomputer software business resources are ample, while the premiums on short time-to-market, on the "tightness" and on the reliability of the produced code are all high. Under these circumstances, adding extra people to projects is even less helpful than it was indicated in Fred Brook's classic "Mythical Man-Month". There is also a shortage of "star" quality programmers.

As to the solutions, on the personnel front we are expending a lot of effort to search out talent in the US and internationally. We also organized an internal training course for new hires. However, these measures will just let us continue to grow but no improvement in the rate of growth can be expected.

I think that the remedy to the productivity question with the greatest potential will come from a long-lasting, steady, and inexorable effort of making small incremental improvements to every facet of the programming process which 1) is easily mechanizable, and 2) recurs at a reasonable frequency which can be lower and lower as progress is made. I think of the Japanese approach to optimizing production lines as the pattern to imitate.

In Hitachi's automated factory for assembling VCR chassis general purpose and special purpose robots alternate with a few manual assembly workers. The interesting point is that there was no all-encompassing uniform vision: Hitachi engineers thought that general purpose robots were too slow and too expensive for simpler tasks such as dressing shafts with lubricant or to slip a washer in its place. At the opposite extreme, a drive band which had to run in several spatial planes was best handled by human workers, even though, in principle, a robot could have been built to do the job, but such a robot would have cost too much in terms of capital, development time, and maintenance. In the middle of the task complexity spectrum general purpose robots were used. The product design was altered by the engineers to make the robots' work easier. For example, they made sure that the robots can firmly grasp the part, that the move of the robot arm through a simple arc is unimpeded, and so on. The actual product design could not be simplified because it was constrained by competitive requirements. If anything, the increase in production productivity makes it possible to elaborate the design by adding difficult to implement but desirable features. For instance, front loading in VCRs is much more complex mechanically than top loading, yet consumers prefer front loading because of its convenience and because front loading units can be stacked with other hifi equipment. The point here is that simplicity is always relative to the requirements. The requirements always tend to increase in complexity, and there is nothing wrong with that.

These notions have direct parallels in software production. Software complexity should be measured relative to the requirements and can be expected to increase in spite of the designer's best efforts. Frequent, simple tasks should be solved by specialized means (e.g. most languages have a special symbol "+" for addition) and manual methods can be appropriate for the most complex production steps (for example hand compiling the innermost loop.) This is nothing new: the issue is really the proper mix. In manufacturing, as in programming production old shops have obvious problems with reliance on manual labor and on inflexible special purpose machines, while the most modern shops overutilize general purpose equipment and fail to integrate the other approaches properly. Hitachi, in a very pragmatic way, created a new and potent cocktail from the different methods each with particular strengths and weaknesses.

I do not deny that more radical treks in the problem space and in the solution space could also result in major breakthroughs and also wander sometimes whether America has the temperament necessary for the incremental approach I am proposing, that is to pile up the small quantitative changes in search of the qualitative changes. Yet I believe that the payoff from the cumulative improvements will be very large. I also believe that the approach is only medium in cost and low in rick. While the usual rule of thumb is that this would indicate low payoff, I am comfortable with the paradox and blame special cultural, historical, and business circumstances for the unexploited existence of this large opportunity.

I give you a few examples to illustrate the minuscule scopes of the individual improvements to software tools which can be worthwhile to make. During debugging it is often useful to find all references to a variable *foo* in a procedure, Every editor typically has some "find" command which, after the user types in "foo", will scan for and highlight the instances one by one. An improvement, for this purpose, is the ability to point to one instance and see the highlight appear on all instances at once. The ability to discern scopes at the same time would be a bonus. This is hardly automatic programming, not even CASE, just one small step.

As a second example consider that in the programming language we use (C) just as in practically all languages, a procedure can return only a single value as its result. Of course myriad possibilities exist for implementing the return of a second quantity if that is desired: pass a pointer to the location where the second result should go ("call by reference"), return the

second result in a global variable, aggregate the results in a data structure and return the single aggregate value, and so oil. An improvement could be made to the language by adding Mesa's constructors and extractors which remove the asymmetry between the implementations of the first and the second returned value. This would immediately resolve the programmers' quandary by providing a single solution to the problem, and also enable the focus on optimizing whatever the preferred solution may be.

We will need on the order of a thousand such improvements to make an appreciable difference. It will be easy technically. It has not been done before probably because of the following non-technical problems:

- Talented people do not get excited about incremental projects.
- One has to have source access to all the software which is to be modified: editor, compiler, debugger, run-time, etc. Such access is rare.
- The incremental improvements are so small that when uncertain side-effects are also considered, the net result may well be negative instead of positive. Many argue, for example, that the mere existence of something new can be so costly, in terms of training, potential errors, conceptual load on the user, or maintenance, that only profound benefits could justify it.

I consider this last argument profoundly dangerous even while often true. I will discuss it further in my answer to your question #2 below.

I would also like to list a few areas which are not problems at least for the organization I am working in- marketing, product design, implementation methodology, testing methodology, product reliability, working environment, and management. This is not to say that these activities are either easy or are well understood. For instance, we still can not schedule software development with any accuracy, but we learned to manage the uncertainty and we can expect much smaller absolute errors (even at the same relative error) if productivity can be improved. Similarly, testing is still very unscientific and somewhat uncertain. Again, with sufficient safety factors we can achieve the required reliability at the cost of being very inefficient relative to some ideal. With greater productivity, more built-in tests can be created and testing efficiency call improve. So I see productivity as the "cash" of software production which can be spent in many ways to purchase benefits where they are the most needed: faster time to market, more reliability of the product, or even greater product performance. We get this last result by iterating and tuning the design and implementation when the higher productivity makes that affordable.

What do you see as the most critical problem that industry and the nation have with current software production, and what solutions do you suggest?

Under duress, I would venture the guess that for mission-critical applications the greatest problem is reliability, and for all others the cost of development, especially the opportunity costs of lengthy development.

I can make a second-order point, however. Let us call the solution to the nation's software problems the "next generation tools." It is safe to guess that these will be very complex software systems and that they will be developed using more current tools, which connects to my response to Question #.I. Scientists had to measure a lot of atomic weights before the periodic table was discovered. They had to measure an awful lot of spectral frequencies before quantum theory could he developed. We, too, should assiduously do our homework while some of us try for the big leaps (CYC, Automatic Programming, etc.)

I think in the near term, the appearance of an ubiquitous programmer's individual cross development platform would greatly enhance our ability to create more, cheaper, and better quality software. By "ubiquitous" I mean many hundreds of thousands, that is 396 OS/2 based, rather than tens of thousands, that is workstation based systems. By "cross development" I mean that the target system would be separate from, and in general different from the highly standardized and optimized development machine. *[Note: this was in 1989. The system I had in mind then is called Windows NT now in 1994]*

It would be also nice is the ubiquitous platforms used an ubiquitous language as well. Insofar as the user interface to the editor, compiler, debugger, etc. is also a language, possibly larger than a programming language, this commonality is certainly within reach with the advent of the graphical user interfaces, and in particular of the SAA standard. The computer language is a greater problem. Wide acceptance can be ensured, in general, by two means: by the fiat of a powerful organization, as was the case for Ada, or by overwhelming success in the marketplace of business and academe. Here Lotus 1-2-3 comes to mind as an example for the degree of success which might be necessary; in languages C came the closest to this. However, C and even C++, did not make a major effort to address a wider user base, in fact takes justified pride of the substantial accomplishments in software technology which sprung from the focus on narrow goals.

I am a believer in a major heresy. I believe that PL/I *had the right idea*, if at the wrong time. A leading programming language should try to satisfy a very wide range of users, for example those using COBOL (forms, pictures, decimal arithmetic, mass storage access), C++ (object oriented programming. C efficiency, bit level access), SNOBOL (sting processing, pattern matching), ADA (type checking, interface checking, exception handling) FORTRAN (math libraries), and even Assembly language (super high efficiency, complete access to facilities.' Let me just respond superficially to the most obvious objections.

1. PL/I is terrible; PL/I was designed to have everything; Ergo wanting everything is terrible, Answer: What is your complaint about PL/I? X? So you would prefer to have some x in -X and PL/I does not satisfy your desire? So you want more which is also what I want.

2. But what if not having things in my language is the essence of the benefit I am seeking. How can you satisfy that? Answer: I can not. But in fact people are seldom attracted to the lack of things as such. Most people want the safety, the ease of learning, efficiency, availability, and the low cost which are easily (and usually) obtained by the expedient of not having things. However, the same benefits can be also obtained in other ways as well, albeit at a radically higher capital cost. Again, the point is that simplicity should be just one means to some end, not the end in itself. And even giant capital costs need not be a burden to the end user.

3. You can not learn a monster language. Answer: A large piece of software is much more complex in terms of number of identifiers, number of operators, or the number of lines of documentation than a complex programming language, If some elaboration of the smaller part - the language - benefits the bigger part - the software being written in the language - we have a good trade-off.

My feeling is that a small trend of enhancing the most promising language - C - has already started with the growing popularity of C++. I believe that this trend will continue with even larger supersets of C appearing and winning in the marketplace, I think we should encourage this trend, promote a rapprochement between the C people and the data processing professionals, and point out the dangers to those who remain locked into Ada.

David Talbot

Software 2000 - What changes; what stays the same?

Points for discussion

Introduction - What changes in the software world, what stays the same and for both, perhaps, why and what's new?

What stays the same - Enterprises in all sectors of the developed economy have invested in billions of lines of code for so called Management Information Systems which, for the most part, provide neither "management" nor "information" but offer essentially systems of clerical automation. Billing is perhaps a good example with, at a guess, in the order of a billion lines invested in this application alone. Systems of this class are fundamental to the running of the enterprise. They will not go away but will need to evolve in response to possibly two driving forces.

The first of these forces is the need for the enterprise to stay "competitive" in the widest of senses i.e. to respond to changes in the environment and to spot new opportunities to satisfy its customers. In this context the paradox is that these systems are both fundamental to the well being of the enterprise (no billing - no cash) and at the same time their "rigidity" threatens the ability to adapt and exploit. Speed of change and the globalisation of transactions are two of the key issues that need to be managed and within which these huge bodies of software need to evolve.

Overall, "timeliness" and "service" are critical issues and software has to respond to both of these imperatives and on both counts is regarded by its users as failing.

Current approaches to software development are seen as too cumbersome and time consuming - many now push for a "revolution" rather than the incremental improvement currently offered and pursued. Time, as is constantly pointed out, is ultimately "our only truly limited resource".

Service, in the sense of satisfying the customer is also an issue and in this context the trend is towards more demanding customers whether external "end-users" - often the man in the street - or corporate users of software based services. Apart from the usual and general demands for higher quality and more friendly services they seek:

- more timely information about services (the "product")
- more control over the services they use
- more access to services especially information of all sorts
- usable interfaces

The second driving force, is as usual, "technical". In this regard the move from monolithic to more open, distributed and heterogeneous systems is both an economic response, - they are cheaper (at least in terms of basic underlying components), and they offer more flexibility in terms of a better match between the system and the organisation which it serves.

A further "technical" aspect is a recognition that most of these classical software systems have done little to help solve real management problems i.e. problems that involve reasoning about real world situations involving data that is incomplete, contraditory, time dependent, etc. It is possible that this has led to the perceived slowdown in the rate of growth (and margins) of traditional software applications. For these classes of application high levels of saturation now appear to have been achieved and, as a consequence, for this type of system, the main efforts are concerned with "evolving" so called "legacy systems" as opposed to providing new applications. A real need and a new opportunity arises for applications that give new levels of management support - "crises management" whether managing flood control in the Bordeaux basin or speculation against a currency are good but possibly extreme examples.

Some implications and opportunities - For software systems development we have the usual quest for the "philosopher's stone" - new development paradigms; re-use at last (with its associated legal issues); does O-O solve all problems(?); configuration tools and large scale components (part of reuse), etc.

For those struggling with 20 years of code they look for - migration and middleware support; reengineering tools; support for the design and implementation of distributed functionality, etc.

For better end-user services - more natural, better and customisable multimodal forms of interaction - speech, gesture, natural language (sometimes multilingual) dialogues, etc. are needed.

For data swamped managers - combinations of the more tried and promising of the "exotic" software based technologies, their integration seamlessly with major data sources and applications that convert the data flood into manageable information.

What is new - Software is embedded everywhere. This is not so much new but likely to explode with the edge of the explosion led by consumer electronics, games and the need to help people change jobs and roles with a frequency not previously encountered.

In all of these applications issues of dependability and user interaction arise.

Dependability already confronts those providing software driven control systems of all types; from all forms of transportation systems to those concerned with industrial and environmental control. Here the driving forces are usually safety and other legislative pressures. However, dependability from an economic and competitive perspective faces those who are putting increasing bodies of software into "traditional" products such as washing machines, the electric shaver right up to the TV set which is now a major software driven distributed system in its own right and which, in turn, will become an entry point to the "information highway".

Product and service "usability" has already become a key competitive feature. "Smart" consumer products, instant personal information and communication devices and services "plugged" into the much discussed global "information highway/space" will only be as good as the level and ease of interaction offered to the user. Training and education on demand will be only as effective as the "realism/virtual reality" presented to the trainee.

With grateful thanks to insights arising from a recent Industrial User Panel meeting

Tony Temple

We are in a period of accelerating change for developers of computer software. These changes are in all areas of our business: interfaces to the end user, the technology for development, the type of function we can provide, the computing infrastructure we must support, the commercial environment in which we operate. In this short paper I will touch on what I see as the major topics in each of these areas, but before doing so I will state a number of key assumptions I am making.

- This rate of change will continue to accelerate through the '90s
- The major factors driving software changes will be hardware and infrastructure development.
- The bulk of the increased hardware computing capacity will be exploited in the user domain, by more sophisticated end-user interfaces, and by distributing function to individual workstations.
- User departments control their own IT destiny, and in particular control IT investment. This is broadly true today, and will become more so over the coming decade.

Technology for development

- Development capability is moving increasingly closer to the end user. 4GLs, Visual development and Wizard style assistance allow new types of non traditional programmer. Construction kits make programming more accessible. Some sophisticated (and interested) end users are developing their own applications using spreadsheets or databases as the vehicle.
- Standardisation of interfaces among the major suppliers has been a theme of the last few years, and we can expect that consumer pressure will continue to select in favour of suppliers with open interfaces.
- Automatic generation of code from specifications is becoming more feasible, and we should expect to see some actual examples of this commercially in the next few years.
- Object oriented technology currently allows the development of libraries of component software. This provides an effective mechanism for writing code once, and then reusing it where appropriate.
- Historically the size of effective software development projects has been limited by the complexity that can be handled by a single development team. As development becomes more a matter of socketing together a number of already working objects using standard interfaces, it will be possible to design larger projects.

Infrastructure

The system software base is converging towards a graphical object oriented user interface, and providing more standard platform services to applications. With this and with the availability of cross system development tools, the application developer is increasingly protected from the idiosyncrasies of the system software on which he is running. The operating system is becoming irrelevant to the end user.

The hardware base is currently changing. Some old hardware platforms are converging and some new platforms are emerging. New hardware platforms will provide emulation, so changes should be transparent to software applications. Anticipated hardware performance enhancements will make new applications possible - particularly compute intensive user interface technologies. Compute intensive artificial intelligence applications will become more feasible on the majority of user workstations.

Network technology will leap forward dramatically both in performance and cost. Increased bandwidth will allow networking of isocronous multimedia applications. Distributed computing capability (either in traditional enterprise networks, or between enterprises, or to mobile users, or even down to "set-top" systems in people's homes) will mandate a new breed of applications. The much discussed "Information Highway" will become a reality during this decade.

Legacy systems will continue to be supported in traditional organisations. These systems may be too big and complex to amend rapidly, or there may be no current justification for re-engineering them. Client/Server front-ends for these legacy systems is an emerging software market.

Hardware advances such as flat panel displays, low power systems, more efficient batteries, etc. will change the way computers are used. A major software opportunity will arise when people leave their computers on permanently. Users will begin to lose awareness of the underlying storage hierarchy, for example they may want to assume that any changes made are automatically saved and they will want an "Undo" capability to reverse errors. More research is needed to understand the most appropriate user model.

Functionality

An increasingly sophisticated marketplace in conjunction with the changes discussed above will create new requirements for applications. This is a broad topic, but initial thoughts include:

- Telecommuting.
- Home transactions such as shopping, banking, mail, etc.
- Leisure applications such as interactive networked games, network
- Distribution of films, books, etc.
- Workgroup applications that will allow network interaction of humans who are separated by space and time zone.
- Mobile computing requires applications that support intermittent wireless network connection.
- Artificial intelligence (expert systems, data mining, optimisation, etc.) will be increasingly feasible as costs decrease, performance improves and widespread networking is available. More immediately, I believe that Intelligent Agents which have been available for some while on the Apple Macintosh will be developed for other computing environments (for example a networked agent could scan the Information Highway for relevant information).
- As general purpose microprocessors are used in more and more products (cars, consumer white goods, "set tops", etc.) specialised software applications will be needed for these also.

Commercial Implications

Improved development methods will allow either user support staff or brought-in consultants to build custom vertical applications to meet business needs both cheaply and quickly. It seems likely that such integrators will build run-time environments with little or no licence charges - all the price will be up-front.

A few companies are now emerging who specialise in developing or buying software objects, and who will sell the resultant object libraries to integrators.

Prices for software componentry are dropping, and will continue to drop. Eventually basic level application componentry will be provided almost exclusively pre-packaged with system software at very low prices. More advanced applications will be available at a higher price, but

even here competitive pressures will continue to drive prices down. Only specialised niche products (with a restricted marketplace) will be able to command high prices.

Distribution and charging for software have become commercial issues. Network distribution of software is possible now, but how can it be charged for and policed? Should we be investigating payment by usage?

All of this demands a new approach for the traditional software developers. There is considerable opportunity for software development over the next decade, but not all of the existing players will survive these challenges.

End user interface

The challenge to the software industry is to make computing accessible to as wide an audience as possible. This audience will include novices (computer illiterates), who require simplified interfaces and fewer choices, to experts who feel ill at ease unless they are using a command line interface.

New technologies (large high definition screens, voice, pen, scanning, etc.) give new opportunities and new challenges for creating an effective end-user environment. Increasing system performance means that a greater percentage of system resources can be spent providing this environment.

In order to simplify our job as developers, and to provide consistency to our users the software industry needs to agree and implement common standards for the End User Interface. CUA is IBM's GUI standard. Apple and the COSE consortium have proposed an equivalent standard. We are now seeing the emergence of interface standards via the "Compound Document" paradigm (as implemented in Microsoft's OLE and CIL's OpenDoc). These technologies provide a consistent interface between applications and users, and between applications and each other (even from different vendors) - and will allow integration of new data types (voice, image, video, etc.). The software industry must settle on one (or at worst a small number) of such standards, also a critical body of software must be implemented to this standard.

Within this decade it is unlikely that the primary interface to the computer will change from "glass" to some other method (virtual reality, gloves, direct input to the eye, direct output from eye positioning, direct brain interaction, etc.) so we can expect that the "Compound Document" or similar paradigms will be the major influences on the design of end user interfaces.

Intelligent help systems (help that recognises the sophistication level of the user, help that fires up "Just In Time" when a user is hesitating or has a problem, interactive tutorials, Wizards, etc.) will be a major factor in helping a wider user audience to use the new software. The challenge is to build applications that behave consistently, and which provide sufficient help so that a new user never has to use the manuals.

Mario Tokoro

Issues to be discussed

- coping with openness and distributedness
- coping with heterogeneity (of execution environment)
- coping with multi-vendor software competing each other
- enabling (and coping with) local optimization (or adaptation) to users environments.

Position statement: Software as the Society of Spontaneous Agents

In the near future, every home and office will be interconnected by high speed communication networks to enjoy various interactive multimedia services. Computing systems to support such an environment will consist of various software modules developed by different manufacturers and changed from time to time, and dynamically integrated, operating at multiple sites as servers. Such a system shows emergent behaviors which cannot be ascribed to individual software modules, eventually forming a "society" that is analogous to the human society and interleaved with it.

Hence, our research target is the development of technology for creating safe, evolutionarily stable, cohabitating information society. As an evolved notion of "objects", I would like to propose "autonomous agents" which react to inputs according to their situations and individual goals of survival, and "spontaneous agents" which are more active and aggressive. The ideas of agents, in accordance with research in biology, ethology, and theory of evolution, as well as complex systems as their theoretical basis, will contribute to realize "intimate" computer systems and bring about harmonizing information society to humans.

References:

Tokoro, M. "Computational Field Model: Toward a New Computing Model/Methodology for Open Distributed Environment," in the 2nd IEEE Workshop on Future Trends in Distributed Computing Systems,Cairo, 1990.

Tokoro, M., "The Society of Objects," an invited talk presented in the OOPSLA'93 Conference, Washington, D.C., 1993. An edited transcript available as Sony Computer Science Lab Technical Report SCSL-TR-93-018.

Peter Wharton

Introduction

What is the future of the 'Software Industry'? There are many aspects of this that need evaluating. This note is just one of the perspectives that can be taken. (note: *as I have created this note from some previous ones its scope is slightly wider than software however it is I think all relevant if slightly larger than I would have liked!)*

The different industries and functions that now use software is now so wide that it is dangerous to attempt broad generalisations about its evolution although it is true that software in one industry, e.g. games, may well impact another; for example, commercial use through changing interface expectations. This note, however, primarily concentrates on the changing nature of systems in the corporate commercial/business world. In particular, I have viewed things from an 'engineering perspective'; that is how are the 'total systems' constructed, what are the [changing] requirements on these, what are the disruption factors, etc. Further given the central role that software has in all systems its future is inseparable from that of how the total IT systems are going to evolve.

This is discussed under the following subjects; business pull, technology push, engineering mechanisms.

Business and IT

The role of IT in business has been progressively changing for the last thirty years, until it appears we are now reaching the stage where their interactions are so interrelated that it is not sufficient to think in terms of a series of evolutionary stages but to consider what could, to individual businesses, be revolutionary steps.

The classic approach to business and IT was, and often still is:

Business strategy → **Organisation strategy Business process** → **IT support**

That is a business decides on its objectives and its strategy for achieving them, identifies the organisation and processes needed to support this strategy, then produces an IT system to support this. This model has been effective for some time; however, more and more there has been a feed-back loop whereby the capability of the IT support causes different (hopefully more effective) organisation structures and processes to be adopted. This in turn leads to business strategies being developed based on the competitive advantage to be gained from IT. This overall trend has been fully investigated and discussed in, for example, the Corporation In The 1990s (M. F. Scott-Morton, OUP). The results of this trend leads to requirements not only for integration within and between IT systems, but within and between Businesses themselves. The same language we would have used to discuss IT systems is now being used to discuss the business; Enterprise Architectures, Business Re-engineering etc., etc. Out of this are coming demands for groupware, process support systems workflow management via the IT system, these kinds of solutions not only will need to interoperate in the classic open manner, **but by virtue of being the IT envelope within which systems develop, could well be the basis for software systems of the future.**

Technology

So far, in the history of the IT industry, it has been technology driven and although the prime drivers are now the business ones (how to deliver total solutions, the requirements of the end user etc.), technology is still continuing to develop at an almost frightening rate. Some of the technological advances are basically more of the same, e.g. faster and faster processors in smaller and smaller packages, while others introduce new capabilities completely different from

those they supersede. This continuous improvement of technology represents a challenge to the IT industry and the businesses they support. On the one hand, businesses would like stability and protection of existing investment, on the other the desire to achieve some competitive edge over a competitor encourages the early exploitation of new technologies. Many technological advances can be accommodated within existing standards and structures allowing relatively safe exploitation of the latest capability. Obvious examples of this are the continuous evolution of the INTEL chip for PCs; each stage is compatible with the previous one, but with the potential for something new (e.g. the progressive improvement in main store addressing capability). Other technologies while inherently being of a revolutionary nature, initially fit into the existing systems to achieve early exploitation; examples of this are at the user end, the use of simple terminal emulators on window based VGA terminals and at the platform end the use of parallel systems behind existing interfaces such as SQL for databases or operating systems process interfaces. In this latter case, earlier exploitation of technological advances is possible but their full potential may be suppressed. Finally there are technologies which blatantly cannot be hidden within existing systems and will force new interfaces or solutions to be developed (a current example of this is certainly neural systems. However, true exploitation of parallelism is almost certainly in this class as well, as are multimedia workstations).

From the point of view of IT Systems therefore, it is important to recognise those technologies which in some way will disrupt the status quo; this will include those requiring changes to existing standards or methods, those giving the potential for completely different solutions or simply those which although apparently a simple extension of the day, represent too great a change of degree. It is the handling of these disruptive technologies that could be regarded as the key to future Systems/software.

The dominant system factors are clearly both the speed of change and the need for a smooth evolution of installed systems in the exploitation of new emerging technologies. In particular three criteria have been used, these are:

- Organisational Adoptability (how easy is it for an organisation to adopt)
- Community Adoptability (how much does the benefit depend on everyone else adopting it)
- Agent of change (how likely it is to facilitate some new approach or make an existing one obsolete)

The technologies suggested as likely to have the above attributes are :

Semi-conductors, Parallel Processing, Neural Computing, Integration Architectures and Re-use Methods (particularly Object Oriented technology), Distributed Environments, Multimedia, Business Process Support (Groupware)

Engineering mechanisms

While each of the above technologies both individually and in combination is potentially a significant influencer on the future of software I would like to consider two in more detail as they are , I feel , the ones that could change the nature of how systems are constructed and hence the basis of all future software.

Integration architectures & re-use methods

The complexity inherent in most large enterprises IT systems has always required the a structured approach through the use of architectural methods supported by sophisticated development tools for the actual implementation and construction. This requirement is becoming even more demanding as a result of ;

the rapid development of new technologies and the rate of improvement in existing ones. In particular the ubiquitous availability of processing power is allowing completely new systems approaches to be taken.

- competitive pressures both in terms of cost and time to market resulting the need for greater component re-use and new approaches to development.
- the need to build on the existing (legacy) systems (rather than replace them) as businesses have become dependant on their IT systems to operate.

Currently various approaches exist that can be used in response to these needs including:

- object oriented technologies
- client server architectures
- integration architectures

All of these must be understood and their impact on CSD products and systems catered for. In particular the object oriented approach is likely (eventually) to the most significant and its concepts must be incorporated into all levels of systems software and supported in application development tools.

Business process support (including groupware)

Exploitation of business process support technology repositions the role of the IT system with respect to the business which it supports. It changes the relationship from one which essentially supplies applications on some appropriate network of platforms to one which provides a co-ordination architecture for the support of staff participating in a nominated set of business processes.

There is a need for an architecture (and supporting products) providing support for the integration of a heterogeneous network of platforms and applications. Ownership (shared or otherwise) of such an architecture will give a leadership role in exploiting the technology. This will need to address:

- integration with, and of, many ICL and external groupware products including: Teamforum, ProcessWise, Lotus Notes.
- integration with platform components including:
    - Persistent Database, Long Transactions,
    - current TPMS and Tuxedo environments.
- integration with UI components such as the surface UI, Multi-Media.
- integration with commodity applications such as Document Handling Systems, Spreadsheets, Planning Systems.
- integration with distribution architecture such as: DAIS, OSI Protocols

The Workplace oriented facilities are being (and will continue to be) desktop oriented. At the Enterprise level however there still appears to be a need to develop a full Enterprise wide process support system and integrate it with the evolving Workplace facilities on the one hand and the existing corporate applications (particularly TP) on the other. This potential relationship with TP makes it a key feature in the evolution of legacy systems and hence of paramount importance to all large corporate exploiters of IT and their suppliers . It should be noted that this relationship is not just one of "handling history" but also of ensuring the new capabilities of process managers and groupware products possess the same levels of integrity that today's OLTP systems have. The ACID properties that apply to today's transactions that last for seconds will need to apply to processes that last for days or weeks.

Conclusion

In conclusion I think this means the continued development of such concepts as 'framework' architectures, the use of models within which standards can be defined, coexist and evolve, and the formalisation of many concepts such as encapsulation, modularity, inheritance, use of early and late binding etc., in such ideas as client-server architectures and object oriented technologies. The exploitation of these capabilities within the context of the business and technology oriented pressures (supported by process managers and groupware products) will begin to dominate how we create the total solution. The introduction of OLTP integrity properties will not only be needed for line of business applications but will also be significant in the integration of existing systems.

Overall I would like to see these being regarded as a **"don't care capability"**; that is methods and tools that don't rely on the exact implementation of the same interface or protocol, and are flexible enough to deal with change in a cost-effective manner. These may appear in many guises; change management architectures, integration mechanisms, reverse engineering tools, etc. In the same way that "cut and paste" has allowed interchange of information across PC applications with no or minimal pre-planned co-operation we need to look for concepts that can be generalised across a globally distributed system allowing full and spontaneous access to and between the applications and information as the user finds the need.

Software futures

We might divide the discussion into those areas where we feel that a fairly confident prediction can be made for the year 2000, and those where we see large (possibly binary) unknowables. I shall arrange my input along this knowable/unknowable axis; although my unknowables may be someone else's obvious prediction, and vice versa.

To me, the easy predictions are broadly about technology per se, and the unknowables are about its take-up in business and society. Technology advances in a fairly regular manner, while its take-up goes in fits and starts, depending on key applications which alter public perceptions. (If desktop publishing had not kept the Mac alive, would Microsoft have ever needed to develop Windows?)

- Hardware will be powerful, interconnected, and embedded in the furniture. Some kind of tablet PC, A4-by-1", will be an inevitable accessory in every briefcase or schoolchild's tote bag. It will be permanently interconnected by cellular network or infrared links within buildings.
- Corporate and National networks will be messy and ubiquitous: there will be a mosaic of competing and overlapping 'standards' at all levels, and somehow most of the time it will work. Details of the network will be almost-hidden by software agent technology, but will occasionally break through unpleasantly.
- There will be a thriving market in software components: Solutions to the problems of advertising, distributing and paying for software components over the net will have evolved. High-value components will be leased on a pay-per-use basis. Standards for interfacing utility components (OLE2 etc.) will not converge, because requirements keep changing.
- Who will dominate the market for software components? Monoliths or small businesses on the net? Generalists or vertical market specialists?
- Will the software industry be a tool and component supplier, with application building in the hands of end-users?
- When will software become an engineering discipline?
- Will software still be driven by the needs of business?
- Will the package/component proportion of business IT solutions continue to grow, or plateau at some point?
- Will the scale and risk of advances commerce (rather than e.g. defence, home applications or education) projects reduce?
- Will object orientation and the net further lower the barriers to entry for software product supply?
- Will the software crisis go on for ever?
- Will Open Systems ever converge?
- Will the balance of cost in application development shift further away from software technology knowledge, towards application knowledge? Or towards user-centred knowledge?
- Will every application have good built-in training/education/help?
- Will we reach a plateau at which user expectations are stably matched with what is supplied? (i.e. where the user's mental model of what the system will do for him is well matched by reality)

- Will multimedia I/O (voice, pen, video) be built into most applications?
- Will the overall success rate of projects improve?
- Will the ratio of business benefit/cost of IT solution improve?
- Will there be earthquakes on the way?
- Will IT procurements still be based on unrealistic, over-ambitious aspirations?
- What core skills will continue to command a high price?

Software industry - question areas

One can envisage two broad scenarios:

(a) where there is some kind of stabilisation around the things we can do predictably with good business benefit, so the industry becomes routinised, less of a high-wire act, more in the hands of users with occasional support from suppliers, or

(b) as some things get easier, people demand more because companies are always striving for distinctive competitive edge; software is by definition where people put the unsolved problems, and is always risky.

Crudely, the issue between scenarios (a) and (b) is: will we always keep inventing new hard problems to solve? Several of the question areas below focus on this dichotomy.

- Business Solutions : Will the ratio of business/cost of IT solutions improve? Will the overall success rate of projects improve? Will many IT procurements still be based on unrealistic aspirations and timescales? What core skills will continue to command a high price?
- Software Engineering Practice : Will software ever become an engineering discipline? Or is it by definition where people put the unsolved problems? Will the software crisis go on for ever?
- Structure of the Industry (a) : Will the software industry become a tool and component supplier, with application building in the hands of end-users? Will the available components become so good that all projects can be done by a small team, or will there always be large difficult projects?
- Structure of the Industry (b) : Who will dominate the market for software components? Monoliths or small businesses on the net? Generalists or vertical market specialists? Will object orientation and the net further lower the barriers to entry for software product supply?
- Style of Applications (a) : Will every application have good built-in training/education/help? Will multimedia I/O (voice, pen, video) be built into most applications? Will applications converge on a few standard "personalities" which most users know and can operate, or will there be constant pressure for innovative interfaces?
- Style of Applications (b) : Will a younger generation, computer-literate from childhood, develop a fluency in IT which leaves our generation open-mouthed and obsolete?
- Standardisation : Will broad standardisation moves converge, or will there always be important new areas where standards have not converged?

Position paper

Rather than jump straight to a ten-year vision of the future, I shall approach it in four gentle stages - first developing a "vision of the person", picking out the key trends of today which are shaping IT, then looking at some near-future developments in the 1-3 year timescale, then the futurology proper, looking at the 4-10 year timescale, and finally discussing some areas where I do not expect change.

Throughout this there runs a theme. It is that the important new applications of IT will be determined by people and organisational issues, not by technology. Advances in technology now come in a fairly predictable steady progression - everyone knows how powerful chips will be in three years' time - but advances in the use of technology are much harder to predict. They depend on many factors, and above all they require an avalanche effect; somebody does something new, which succeeds and catches the imagination, then others copy and expand the original idea. We cannot predict where or when the avalanches will occur.

Directions of change today

- User power continues to grow

  Even ten years ago, users were often in awe of IT, believing that the high priests of the it industry could work miracles and entrusting them with expensive, long-term projects. Now with ubiquitous PCs, users are much more familiar with IT, know what it can do for them, demand that it give measurable business value, and want results fast. The market has broken the monopoly power of IT suppliers, and pressure on IT budgets has further squeezed suppliers; the Open Systems movement is one expression of this increased user power.

- IT becomes a commodity

  For routine "housekeeping" applications the trend towards packaged solutions is inexorable; outsourcing is another manifestation of the commoditisation of this sector. Only for those applications which define a company's competitive edge is there a case for building unique bespoke solutions, and even these are built more and more from standard building blocks - hardware and software. However , the building blocks do not easily fit together.

- IT is used to re-engineer business processes

  Although the BPR movement is to a large extent a re-packaging of old ideas, nevertheless radical redesign of business processes is increasingly needed to survive, and IT is often the key enabler. BPR initiatives include many failures, but enough successes to ensure their continued importance. The rapid improvement of PC and networking tools continues to open up new ways of re-thinking one's business.

- Large projects are difficult

  Advancing technology does not notably diminish the rate of failures in large complex projects; perhaps because our ambitions grow as fast as our capabilities. The three main reasons for project failures identified by Curtis et al (ACM communications, 1988) - the thin spread of application domain knowledge, fluctuating and conflicting requirements, and communication and co-ordination difficulties - look set to remain unsolved for some time. If anything the rate of change of user requirements is accelerating.

Near-future developments (1-3 years)

- Open Systems Ferment

  As users have demanded open systems and suppliers have fallen over themselves to offer them, the situation has resembled a large caravan in the desert; a clamour of shouting, hubbub and activity. The caravan is raising a lot of dust, but is it moving? All the pitfalls of standardisation are there - competing vested interests, massive complexity, and attempts to

fix standards in advance of any proof that they work. In computing terms, it is trying to solve the world, without a project manager. The signs are that convergence and compatibility will not be achieved on a broad front, but will coalesce around a few de facto standards defined by the market A proportion of these will come from Microsoft.

- PC/LAN Based Applications Builders

  From Visual Basic to Powerbuilder, tools which two years ago were only suitable for simpler applications are rapidly maturing. Under pressure of intense competition, they are having to tackle the large-project problems of configuration management, testing, and lifecycle management. Some will succeed. The RAD tool suppliers are succeeding where CASE suppliers failed, and will soon swallow them. In this thriving market, picking the right tool can give very high productivity, and the borderline between package and bespoke solutions will become increasingly blurred.

- Distributed Applications

  The increasing availability of corporate, national and international networks, reaching into the office and the home, makes possible new and more effective ways of working. These depend not on any major advance in technology, but on fitting together today's technology - messaging, databases, video - in ways that meet a real need. Predicting where this will happen first (in telecommuting, distance learning, entertainment, or IT system development) is impossible. Every new application is a social, human experiment and most will fail. Some will succeed, precipitating local avalanches of "me-too" applications, which will eventually reshape the way we work and play.

The future (4-10 years)

- IT will start to disappear

  IT will increasingly be embodied not in separate devices, such as the PC on the desk, but will become part of the furniture (the desk itself). We will be concerned not that our workstation conforms to Open System standards, but that the washing machine, the car and the TV conform to them.

- The Personal Tablet

  One device which will remain visible, and which every employee will expect as a matter of course on the day he or she joins a company (just is one now expects a phone), is a personal "tablet computer" around the size of an A4 or A5 pad, which serves as PC, telephone, fax, notebook and filing cabinet. Predictable technology trends will have passed a threshold where this tablet is as congenial and convenient as paper; the screen and typeface are of such a quality, it is so easy to browse and annotate, that we are no longer tempted to print every large document in order to read it. The device communicates wherever it is, allowing one to be permanently connected to the corporate network.

  For all possible advantages, this device may actually reach mass markets through one simple benefit - replacing the briefcase. Such tangible personal symbols, like the Walkman, drive the major markets shifts.

- The Corporate IT Platform

  The corporate survivors in this timeframe will have been through one or more successful "bet your company" reengineering exercises. Most of these were not initially successful, and had to be hastily re-jigged in response to the changes they created in patterns of work, employee motivation and so on. This gives a strong emphasis on having a standard corporate IT platform - network, object libraries, tools, interface standards - on which new processes, workflows and applications can be rapidly prototyped and modified. The quality and adaptability of this IT platform is a key corporate differentiator.

- Lowered Barriers to Entry

  Today's thriving market in custom controls for Visual basic gives a glimpse of the shape of tomorrow's package software markets. Object Orientation and the Net together will mean

that any clever hacker (or business user) who spots a business need can configure objects to meet it (out of other objects, to be used by other objects) and start selling them over the net; perhaps on a pay-per-use basis. Sophisticated search facilities will bring his clients to him without vast advertising budgets, and they can check out his products on customer satisfaction newsgroups.

So the barriers to entry as a software package supplier will be greatly reduced. This may produce a period of great unpredictability in package markets, as established players find their monolithic products encircled by agile object-oriented competitors, battles for brand awareness ensure in the new media, middlemen and brokers emerge. However, the end result of improved market mechanisms should be greatly improved products.

- Earthquakes

  We should not imagine just a steady stream of improvement; there may be a major setbacks. Many of today's IT procurements are over-ambitious on several fronts; they often result in systems which barely do the job they were built for, and are very thin on necessary "non-functional" attributes, such as safety, security, and robustness.

  Over-ambitious, under engineered IT systems are used by a society which is increasingly dependant on them for personal safety, security and prosperity. Life-critical or business-critical systems are increasingly computer-centred. This is a dangerous mixture, like an increasing subterranean stress along a fault line. The stresses may suddenly release.

  Some time in the next few years there may be one or more headline-level computer-caused disasters - major loss of life, or collapse of businesses, or breaches of personal privacy which capture international air time, lead to mega-lawsuits, government enquiries and the like. The result of this could be some ill-considered retrenchment - an embargo on using IT in certain ways defined by IT-illiterate lawyers - which is major hindrance to all the beneficial use of IT.

Some things never change

- Large Projects will still be Difficult

  In spite of improvements in application-building tools at all levels, there will still be a proportion of large IT endeavours which stretch our capabilities to the limit; because businesses will always want some more ambitious IT system to give them competitive edge. The difficulties which beset those projects now (lack of application knowledge, fluctuating and conflicting requirements, communication and co-ordination problems) will be just as prevalent in 10 years' time.

- Open Systems will still be Converging

  As rapidly as convergence is achieved in one area (whether de facto or de jure) new areas will emerge where users want new standards. This reflects not just technological progress, but a fundamental dynamic of the IT supply industry; if all that you supply is defined by agreed standards, you have no competitive edge except price - a vulnerable position to be in. Every supplier will be striving to be conformant enough to be on the bidders' lists, yet to have enough proprietary added-value for good profit margins (and who knows, maybe a bit of proprietary lock-in of clients).

- We will still be Optimists

  The IT community, within both supplier and user organisations, will continue to attract people who are inspired by the vision of what IT might do, rather than those under-impressed by what it actually does; whose first appraisal is "what I could do given a few good people" rather than "what this team can do within its constraints" and who believe in progress more than in obstacles. Long may it remain so.

Bill Wulf

The impact of IT on scholars in the humanities

A great many forces are acting on the software industry, for example intellectual property concerns, an evolving merger between computing and entertainment, and so on. Among the greatest of these forces, I think, is an enormously expanded and significantly changed telecommunications web -- the "national information infrastructure" as it is called in the US. The existence of this infrastructure will facilitate if not demand information services that we are all probably too myopic to see now.

Over the last four years I have traversed a personal mini-odyssey that I think is highly relevant to the future use of this infrastructure, but in ways that I do not yet see clearly. In particular, four years ago, with typical technological arrogance, I presumed that all humanistic scholars were technophobes, unable and often unwilling to learn to turn on their word processor. I was wrong, and I now believe that:

- information technology will have a larger impact on scholarship in the humanities in the next two decades than on the sciences, and
- the uses of information technology in humanistic scholarship is a better model of what "everyman" wants, and we should pay a great deal of attention to how to support their uses.

The process by which I came to these conclusions is a longer story than should be told in a brief white paper. Instead let me say a bit about what we have done about it.

Humanities scholars as user models for "everyman"

We have created an Institute for Advanced Technology in the Humanities (I apologize for the awful name). With the aid of a generous equipment grant from IBM we appoint a limited number of Fellows with interesting scholarly projects and provide them with fairly lavish equipment and personnel support. From their perspective this provides a "jump start" on projects that could be done no other way -- it literally lets them seek answers to questions that they didn't even know how to pose a few years ago. From our perspective it let's us learn how they approach problems and what sort of support they need.

To date we have supported scholars in, among other things, 19th century American history, mediaeval literature, contemporary music composition, archaeology, and 19th century English literature and art. Perhaps the first lesson learned is that, like the sciences, each discipline has its own methodology and sociology -- and not to expect a single "answer" for all humanistic scholarship.

I will try during the workshop to provide more details, but let me just say that, like other converts, I have become a zealot. I have **never** been so excited by the prospects for the impact of information technology!

Appendix C
Working groups

This Appendix lists the participants in each Working Group. For the full expansion of the participants' affiliations, see Appendix A.

Environment

Alessandro Giacolone	ECRC
Brian Gladman	MOD
Cliff Jones	Manchester University
Andrew Herbert	ANSA
Jessica Litman	Wayne State University
Chris Phoenix	ICL
David Talbot	Esprit
Bill Wulf	University of Virginia

Business

Les Belady	MERL
Remi Bourgonjon	Philips
Mike Braude	Gartner Group
Mike Lesk	Bellcore
Gill Ringland	ICL
Robert Worden	Logica

Technology

Hervé Gallaire	Rank Xerox
Gilles Kahn	INRIA
Bill O'Riordan	ICL
Brian Randell	Newcastle University
Charles Simonyi	Microsoft
Tony Temple	IBM
Mario Tokoro	Keio University and SONY

Appendix D

Acronyms

ACCESS	Microsoft's database product
ACID	Properties required by a resource manager: Atomicity, Consistency, Isolation and Durability
ACM	Association of Computer Manufacturers
ADA	A programming language widely used in defense systems
AI	Artificial Intelligence
ANSA	Advanced Network System Architecture, developed by the UK based Architecture Projects Management (APM) Ltd.
API	Application programming interface
ASCAP	American Society of Composers, Authors and Publishers
ATM	Asynchronous Transfer Mode
BCS	British Computer Society
BPR	Business Process Reengineering
BSI	British Standards Institute
BT	British Telecom
CADES	ICL's design repository for its VME operating system
CAP	Computer Analyists and Programmers
CERN	Centre for European Research into Nucleonics, Geneva, Switzerland
CGE	Compagnie Generale d'Electricite SA
CHORUS	Unix microkernel from Chorus Systemes SA
CISE	Computer and Information Science and Engineeering
CMU	Carnegie Mellon University
COSE	Common Open System Environment, a consortium of systems vendors developing UNIX standards
CSCW	Computer Supported Cooperative Work
CSD	(ICL's) Corporate Servers Division
CYC	a project at MCC, Austin to define an intelligent machine
DAIS	An object oriented application development and SI toolkit from ICL, based on ANSA technology
DIALOG	One of the leading on-line database hosts
DSP	Digital Signal Processing
DTI	(UK) Department of Trade and Industry
DTP	Desktop publishing
ECRC	European Computer Research Centre
ELIPSYS/ECLIPSE	Parallel processing architecture from ECRC
FTP	File Transfer Protocol
G7	Group of 7 highly industrialised countries
GDP	Gross Domestic Product
GSI	Generale de Service en Informatique SA
GUI	Graphical User Interface
HCI	Human-Computer Interface
HIPPI	High Performance Parallel Interface
ICP	(IBM's) Interconnect Controller Program

IDC	International Data Corp.
IEE	(UK) Institute of Electrical Engineers
IEEE	(US) Institute of Electrical and Electronic Engineers
IKBS	Intelligent Knowledge Based Systems
ILLIAC-4	an early computer
ILOG	ILOG SA - a French software house
INRIA	(French) Institut National de Recherche en Informatique et Automatique
IP	Intentional Programming
IPR	Intellectual Property Rights
ITALTEL	Societa Italiana Telecommunicazioni SpA
JFIT	Joint Framework for IT
LEO	an early computer
MCC	Microelectronics Computing Corp. Austin, Texas
MERL	Mitsubishi Electronics Research Laboratory
MIMD	Multiple instruction multiple data
MIP	Million instructions per second
MOTIF	A standard Graphical User Interface from the Open Software Foundation (OSF)
NAFTA	North American Free Trade Area
NIC	Newly Industrialised Country
NL	Natural Language
NRC	(US) National Research Council
NSF	(US) National Science Foundation
NT	(Microsoft's) New Technology, next generation operating system
OLE	(Microsoft's) Object Linking and Embedding
OO	Object oriented
OOA	Object oriented analysis
OUP	Oxford University Press
PARC	(Xerox's) Palo Alto Research Centre in Stanford, California, USA
PCTE	Portable Common Tools Enviroment
PDA	Personal Digital Assistant
PIMS	a database of several thousand companies with their business parameters "Profit Impact of Marketing Strategy"
PL/I	a general purpose programming language
POSC	Petrotechnical Open Software Corporation, a consortium to define open systems standards, mase up of the major petrochemical companies
RAD	Rapid Application Development
RC	Regnecentralen, the former Danish computer company
RISC/CISC	Reduced Instruction Set Computing/Complex Instruction Set Computing - microprocessor architectures
SE	Software engineering
SERC	(UK) Science & Engineering Research Council
SIMULA-67	a programming language
SNOBOL	an early programming language
TAURUS	a failed project to provide a paperless trading system for the London Stock Exchange
THERAC	Therac-25 was a radiotherapy machine that malfunctioned and killed people.
TP	Transaction processing
TPMS	TP management system
TSS	Time Sharing System

UCL	University College London
VCR	Video cassette recorder
VDM	Vienna Development Method - a formal method
VIM	(Lotus Corp's) Vendor Independent Message
VME	(ICL's) Virtual Machine Environment operating system
WAIS	Wide Area Information Server

To order
Quote reference number P4265 *to:*
Carole Roberti, D2D, Cavendish Road, Stevenage SG1 2DY, UK
email c.roberti@ste 14.win.icl.co.uk
D2D is the contract manufacturing subsidiary of ICL